TRACTOGRAPHIE PAR APPRENTISSAGE PAR RENFORCEMENT

par

Antoine Théberge

Mémoire présenté au Département d'informatique
en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, 7 avril 2021

Le 7 avril 2021

Le jury a accepté le mémoire de Antoine Théberge dans sa version finale

Membres du jury

Professeur Pierre-Marc Jodoin
Directeur
Département d'Informatique

Professeur Maxime Descoteaux
Codirecteur
Département d'Informatique

Christian Desrosiers
Professeur
Membre externe
Département Département de génie logiciel et des TI
École de technologie supérieure

Laurent Petit
Maître de conférences
Membre externe
Département Centre Broca Nouvelle-Aquitaine
CNRS/Université de Bordeaux

Professeur Martin Vallières
Président-rapporteur
Département d'Informatique

Sommaire

L'Imagerie par Résonance Magnétique de diffusion (IRMd) est présentement la seule technique non-invasive permettant d'étudier la structure de la matière blanche dans le cerveau humain. L'IRMd permet une reconstruction indirecte de la matière blanche grâce à la modélisation du mouvement de l'eau et la tractographie.

La tractographie a été décrite comme un problème mal-posé ; malgré les nombreux algorithmes développés, il demeure très difficile d'évaluer la connectivité globale du cerveau selon des actions basées sur des informations locales.

Motivées par l'explosion des performances de l'apprentissage profond supervisé, des tentatives ont été faites afin d'utiliser cet outil pour concevoir des algorithmes de tractographie exempts des problèmes affligeant la tractographie classique. Cependant ces méthodes, apprenant de données provenant des algorithmes classiques, sont à ce jour vouées à reproduire les même erreurs.

Parallèlement, l'apprentissage profond par renforcement a récemment connu des avancées extraordinaires menant à des percées telles que AlphaGo. L'apprentissage profond par renforcement, par opposition à l'apprentissage profond supervisé, permet à l'algorithme d'apprendre par exploration, ne requérant qu'un signal récompensant les actions adéquates de l'agent apprenant.

Dans ce mémoire, nous aborderons la possibilité d'apprendre à un algorithme d'apprentissage profond par renforcement à reconstruire les chemins de la matière blanche sans avoir recourt à des données biaisées par les algorithmes classiques. Nous poserons le problème de la tractographie dans le contexte de l'apprentissage par renforcement, décrirons les pièges à éviter lors de la conception d'un tel algorithme, puis proposerons une méthode permettant d'obtenir des résultats compétitifs aux algorithmes de tractographie existants.

Sommaire

Mots-clés: tractographie ; apprentissage profond ; apprentissage par renforcement ; imagerie par résonance magnétique de diffusion

Remerciements

J'aimerais tout d'abord remercier mes deux superviseurs, Pierre-Marc Jodoin et Maxime Descoteaux, d'avoir cru en moi et de m'avoir permis d'entreprendre cette maîtrise décidée à la dernière minute. Je crois que personne ne savait l'ampleur du projet dans lequel nous nous embarquions, et je les remercie de leur support, leurs conseils et leurs perspicacité tout au long de ces deux années.

Je remercie aussi tous mes collègues des laboratoires SCIL et VITAL pour m'avoir supporté et motivé lors de ces deux années de maîtrise, pour leur renforcement négatif face à mes tractogrammes pourris et leurs blagues face à l'apprentissage par renforcement. Comme quoi, finalement, ça fonctionne peut-être l'apprentissage par renforcement !

Je remercie Froduald Kabanza de m'avoir permis d'explorer l'apprentissage par renforcement et la recherche à travers les projets de fin de baccalauréat.

Je remercie l'Université de Sherbrooke, leur concours et journée "Direction Recherche" (destiné principalement aux étudiants au baccalauréat dans une autre université que l'Université de Sherbrooke) et Gabriel Girard, croisé par hasard lors de cette journée, de m'avoir permis de rencontrer Pierre-Marc Jodoin et ainsi de commencer cet extraordinaire processus qui aura duré deux ans.

Finalement, j'aimerais remercier Consensus, la cohorte 2017-2020 du baccalauréat en informatique, de m'avoir accueilli et permis de passer de merveilleux moments en leur compagnie.

—Antoine

Abréviations

BST Tractographie spécifique par faisceaux

CSD Déconvolution sphérique contrainte

DDPG Gradient de politique déterministe profond

dODF Fonction de distribution d'orientations de diffusion

DQN Réseau Q profond

DTI Imagerie par tenseur de diffusion

DWI Image(rie) pondérée par diffusion

FA Anisotropie fractionelle

FCNN Réseau de neurones à propagation avant

fODF Fonction de distribution d'orientations de fibres

HARDI Imagerie de diffusion à haute résolution angulaire

HCP Projet connectome humain

IB Faisceau invalide

IC Connection invalide

IRM Imagerie par résonance magnétique

IRMd Imagerie par résonance magnétique de diffusion

LCS Liquide cérébrospinal

MB Matière blanche

MG Matière grise

NC Non-connections

Abréviations

ODF Fonction de distribution d'orientations

OL Chevauchement

OR Dépassement

PDM Processus de Décision Markovien

PDMPO Processus de Décision Markovien Partiellement Observable

PFT Tractographie par filtrage de particules

PPO Optimisation proximale de politique

RVB Rouge-vert-bleu

SDT Transformée de déconvolution sphérique

SAC Soft Acteur-critique

SET Tractographie améliorée par surfaces

SH Harmoniques sphériques

TD3 Gradient de politique déterministe jumelé et différé

TFR Transformée de Funk-Radon

TRPO Optimisation de politiques par région de confiance

VB Faisceau valide

VC Connection valide

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

The revolution will not be supervised

—Yann Lecun

Introduction

L'imagerie par résonance magnétique (IRM) est un outil extrêmement puissant permettant d'obtenir des images anatomiques à contrastes élevés de façon totalement non-invasive. L'IRM de diffusion (IRMd), obtenant ses contrastes du mouvement des molécules d'eau présentes dans les tissus humain, permet en plus d'inférer la structure hautement organisée de la matière blanche du cerveau humain à travers un processus appelé *tractographie* [16, 18, 91].

L'étude de la structure de la matière blanche, sans avoir recours à la dissection *ex-vivo* de celle-ci, a plusieurs usages potentiels. Par exemple, lorsque couplée avec l'IRM fonctionnelle (fIRM), la tractographie peut informer sur l'activation des régions du cerveau humain et la connectivité physique entre elles [22]. Par elle-même, la tractographie peut être utilisée dans la planification de neurochirurgies [140] ou pour produire des résultats quantitatifs lors de l'étude de maladies neurodégénératives [67]. On peut dire que la tractographie permet d'effectuer une dissection virtuelle de la matière blanche [37].

La tractographie peut reconstruire la matière blanche en la parcourant grâce à des informations locales sur la directionalité de celle-ci. Par contre, la haute complexité de la structure du cerveau implique qu'un algorithme de tractographie doit sans cesse prendre des décisions sur la direction à prendre, alors que très peu d'information pouvant aider est disponible à l'algorithme. Pire encore, la destination est souvent inconnue.

De ce fait, les algorithmes de tractographie souffrent de plusieurs problèmes à ce jour non résolus, particulièrement lorsque plusieurs faisceaux se croisent ou passent proche les uns des autres [114]. Des algorithmes de tractographie par apprentissage automatique ont été proposés afin de palier à ces problèmes grâce à leur expressivité

extrêmement puissante, mais ceux-ci demeurent dépendants d'une vérité terrain qui est très difficile à obtenir, ou biaisée par les algorithmes "classiques" [106].

L'apprentissage par renforcement profond est une forme d'apprentissage automatique où il n'est pas nécessaire d'avoir accès à une vérité terrain. Plutôt, l'agent apprend de façon autonome, par un processus ressemblant à l'essai-erreur, en agissant dans un environnement. À chaque action posée par l'agent, celui-ci reçoit une récompense, une évaluation de l'action posée, qu'il tentera de maximiser en adaptant sa stratégie.

Dans ce mémoire, nous présentons la tractographie par apprentissage par renforcement. Nous espérons palier aux problèmes affligeant la tractographie par apprentissage automatique en évitant d'avoir recourt à la vérité terrain. Le premier chapitre fera un bref résumé de l'apprentissage supervisé, plus particulièrement des réseau de neurones, de leur définition et de leur processus d'entraînement. Le deuxième chapitre abordera plutôt l'apprentissage par renforcement, son contexte, quelques algorithmes "classiques" afin d'introduire des concepts clés, puis finalement quelques algorithmes d'apprentissage par renforcement profond. Le troisième chapitre apportera un aperçu de la tractographie, en décrivant d'abord les méthodes d'acquisition des données de diffusion, puis une description du contexte de la tractographie et les différents algorithmes proposés, ainsi que les problèmes auxquels ceux-ci font face. Finalement, le contexte de la tractographie par apprentissage par renforcement sera proposé, avec ses variantes possibles ainsi que les résultats obtenus.

Chapitre 1

Réseaux de neurones

Introduits en 1958 [146], les réseaux de neurones ont explosé en popularité suite au succès d'une variante, le réseau de neurones convolutif, dans le concours ImageNet LSVRC-2010 [80], en 2012. Depuis, ceux-ci sont utilisés dans le domaine des jeux [126], le diagnostique médical [1], la reconnaissance vocale [53] et bien plus.

Les réseaux de neurones entrent dans la grande famille de l'apprentissage automatique, dont le but est d'entraîner un modèle permettant de prédire une cible selon une donnée en entrée et les paramètres de ce modèle. Plusieurs types de modèles existent, tels que la régression linéaire [12], les machines à vecteurs de support [27], les méthodes à noyaux [12], et bien d'autres, mais nous nous concentrerons sur les réseaux de neurones. Ceux-ci ont démontré ces dernières années qu'ils sont capables de reconnaître des motifs dans les données d'entraînement afin de généraliser à de nouvelles données sans avoir recours à des extracteurs de caractéristiques conçus expressément pour les données en entrée du modèle.

Dans cette section, nous définirons d'abord les réseaux de neurones : nous aborderons leur définition, et comment l'information se propage à travers ceux-ci afin de faire une prédiction. Nous verrons ensuite le contexte de l'apprentissage supervisé, et comment définir une fonction d'erreur permettant d'évaluer les performances des réseaux de neurones. Finalement, nous définirons comment se servir de la fonction d'erreur afin d'entraîner le réseau de neurones.

1.1 Modèles et réseaux de neurones

Commençons par définir le contexte : nous souhaitons avoir un modèle, une fonction paramétrable permettant de prédire une cible selon une donnée en entrée, prenant la forme

$$y_w(x) = t, \tag{1.1}$$

où $x \in \mathbb{R}^I$ est le vecteur d'entrée, $w \in \mathbb{R}^J$ les paramètres de la fonction et $t \in \mathbb{R}^L$ la cible à atteindre.

Dans un contexte général, il pourrait être utile, par exemple, de pouvoir prédire le prix d'une maison selon des caractéristiques données. x serait alors un vecteur contenant plusieurs informations sur la maison, telle que la date de construction, la grandeur du terrain, etc. et t serait son prix en dollars. Un autre exemple pourrait être un modèle permettant de classifier des images de chats, de chiens et de canards. x serait une image et t un vecteur 1-parmi-N de taille 3, où chaque entrée correspond à une classe (chien, chat ou canard), sa valeur étant 0 ou 1 selon la classe de l'image.

Les modèles linéaires sont basés sur une combinaison de fonctions de base $\phi(x)$ et prennent la forme

$$y_w(x) = f\Big(\big(\sum_{j=1}^{J} w_j \phi_j(x)\big) + w_0\Big), \tag{1.2}$$

où le modèle a J fonctions de bases et paramètres, $w_{1..J}$ est le vecteur de poids (ou paramètres), w_0 est le biais et f est une *fonction d'activation* non-linéaire. Dans les prochaines équations, nous traiterons w_0 comme les autres paramètres du modèle en fixant $\phi_0(x) = 1$.

La fonction de base permet au modèle de faire des prédictions même s'il n'y a pas de lien linéaire entre l'entrée et la sortie en transformant l'entrée vers un espace où les données sont linéairement liées à la sortie (dans le cas de la régression) ou séparables (dans le cas de la classification). À titre d'exemple, la régression linéaire utilise la fonction identité comme fonction de base. Alors que les modèles de régression et classification linéaire ont des fonctions de bases définies par l'utilisateur (tel qu'une fonction polynomiale ou radiale) [12], les réseaux de neurones couplent la fonction de base avec les paramètres du réseau afin que le modèle apprenne sa propre fonction de base.

Les réseaux de neurones peuvent être définis comme une suite de transformations séparées par des non-linéaritées, où chaque transformation est une *couche* du réseau. Définissons la couche d'entrée (1) comme étant

$$h_j^{(1)} = f^{(1)}\Big(\sum_{i=0}^{I} w_{ji}^{(1)} x_i^{(0)}\Big),\ j = 0..J, \tag{1.3}$$

où $w_j^{(1)}$ sont les paramètres de la couche d'entrée formant J combinaisons linéaires du vecteur $x^{(0)}$ en entrée. Le résultat du produit des poids et du vecteur d'entrée se nomme l'*activation* et est définie en tant que

$$a_j^{(1)} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i^{(0)},\ j = 0..J. \tag{1.4}$$

Celle-ci est donnée en entrée à une fonction d'activation non-linéaire f, telle que la fonction sigmoïdale ou *tanh*. h_j représente la j-ème sortie de la première couche du réseau.

Suite à cette transformation des données en entrée, nous pouvons prendre la sortie de la couche d'entrée (1) et s'en servir comme entrée à la couche (2) ayant M paramètres :

$$h_k^{(2)} = f^{(2)}\Big(\sum_{j=0}^{M} w_{kj}^{(2)} h_j^{(1)}\Big),\ k = 0..K, \tag{1.5}$$

où f peut être la même ou une autre fonction d'activation. Nous appellerons cette couche intermédiaire une *couche cachée* du modèle. Finalement, définissons la couche (3), la couche de sortie à N éléments en tant que :

$$y_l = f^{(3)}\Big(\sum_{k=0}^{N} w_{lk}^{(3)} h_k^{(2)}\Big),\ l = 0..L, \tag{1.6}$$

où f peut prendre plusieurs formes. Par exemple, pour la régression, f peut simplement être la fonction identité. Pour la classification multiclasse, il est possible d'utiliser la fonction *softmax*

$$f_l = \frac{\exp(x_l)}{\sum_l^L \exp(x_l)}, \tag{1.7}$$

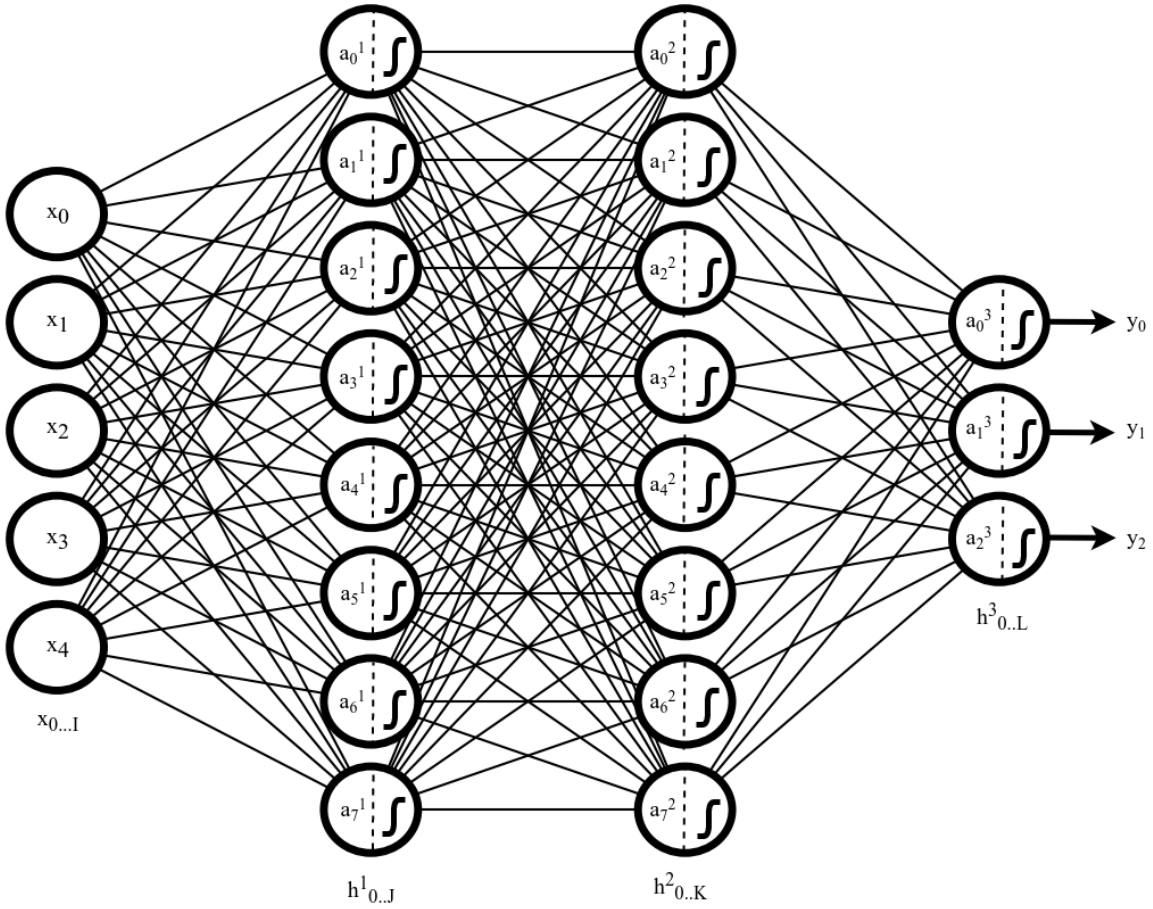

Figure 1.1 – Illustration du réseau à 3 couches, où $I = 5, J = K = 8, L = 3$.

avec L le nombre de classes.

Mis au long, nous pouvons définir le réseau de neurones à quatre couches comme étant la fonction

$$y_l(x, \theta) = f^{(3)}\Big(\sum_{k=0}^{N} w_{lk}^{(3)} f^{(2)}\Big(\sum_{j=0}^{M} w_{kj}^{(2)} f^{(1)}\Big(\sum_{i=0}^{D} w_{ji}^{(1)} x_i^{(0)}\Big)\Big)\Big), \tag{1.8}$$

avec $\theta = (w^{(1)}, w^{(2)}, w^{(3)})$ les vecteurs de poids du réseau.

À partir de cet exemple, il devrait être facile pour le lecteur d'apprécier la qualité récursive des réseaux de neurones, ainsi que d'extrapoler un réseau ayant un nombre différent de couches. Le théorème d'approximation universel indique qu'un réseau de

neurones à seulement une seule couche cachée arbitrairement large permet d'approximer avec précision n'importe quelle fonction continue sur un ensemble compact de l'entrée du réseau [29]. Malgré ce théorème, l'expérience démontre qu'il faut souvent bien plus d'une couche cachée pour obtenir des performances adéquates et l'utilisation de réseaux de plus en plus profonds amènera la révolution de "l'apprentissage profond" que nous connaissons depuis quelques années.

Le type de réseau présenté jusqu'ici et illustré dans la figure 1.1 est dit *à propagation avant* (*feed-forward*), puisque l'information ne se déplace que dans une seule direction, "l'avant" et *entièrement connectés* (*fully-connected neural network*, FCNN), puisque chaque neurone d'une couche est connecté à toutes les neurones de la couche précédente et la couche suivante.

Par contre, plusieurs types de réseaux de neurones existent : les réseaux de neurones convolutifs, par exemple, permettent la réutilisation de poids en appliquant des "filtres" sur les données en entrée (généralement une image) [87]. Différemment, les réseaux récurrents tels que les LSTM [64] permettent d'inculquer une "mémoire" au réseau en le représentant comme graphe cyclique et permettant la réutilisation de l'état interne de celui-ci. Ces derniers ont eu un très grand succès dans les domaines de la traduction [128], de la reconnaissance vocale [54] et de l'annotation d'images [75].

1.2 Apprentissage supervisé

Maintenant que nous avons défini le réseau, nous devons définir la façon d'obtenir les résultats attendus avec ce modèle. L'apprentissage supervisé est une forme alternative à l'apprentissage par renforcement, présenté dans le chapitre suivant, où on suppose l'existence d'une banque D de données d'entraînement prenant la forme :

$$D = \langle (x_1, t_1), (x_2, t_2), ..., (x_N, t_N) \rangle,$$

avec x_i un vecteur de données et t_i la cible à atteindre. Le but de l'entraînement supervisé est de modifier l'ensemble des paramètres $\theta = (w^{(1)}, ..., w^{(N)})$ du modèle afin qu'il puisse prédire la cible t_i à partir de la donnée x_i, dont la figure 1.2 illustre le processus. Pour ce faire, nous aurons besoin d'une mesure indiquant la performance

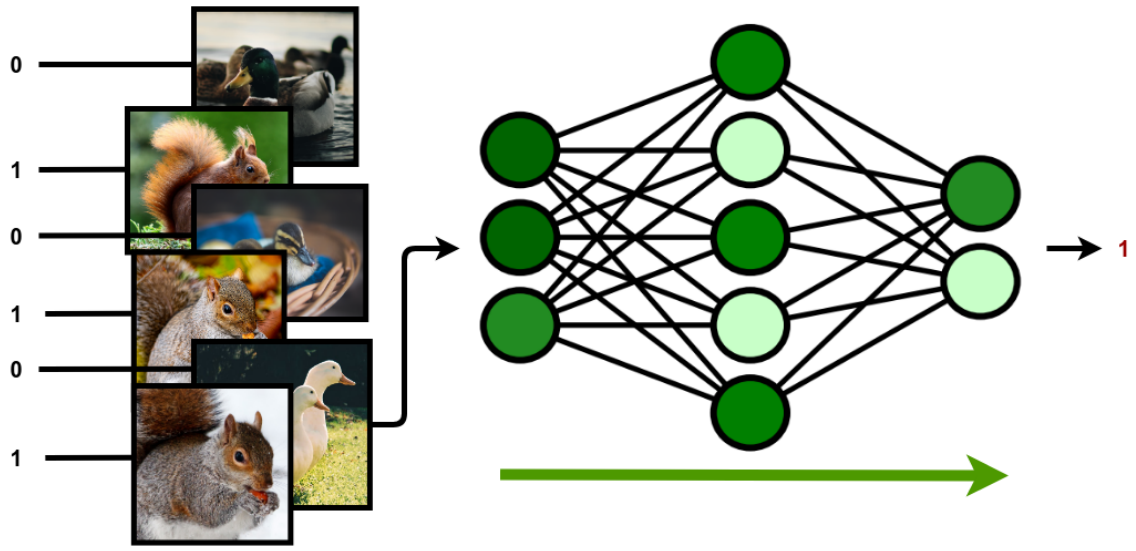

Figure 1.2 – Illustration de la prédiction et de la propagation "avant". Les tons de vert représentent l'activation de chaque neurone.

du modèle, à quel point les prédictions de celui-ci sont correctes. Plus précisément, à travers l'entraînement, nous tenterons de minimiser la fonction d'erreur prenant la forme suivante :

$$E(\theta) = \frac{1}{2}\sum_{n=1}^{N}(\underbrace{y_\theta(x_n)}_{\text{prédiction}} - \underbrace{t_n}_{\text{cible}})^2, \tag{1.9}$$

dans le cas de la régression, et prenant la forme :

$$E(\theta) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{\theta k}(x_n), \tag{1.10}$$

dans le cas de la classification à K classes [12].

Une façon naïve de trouver le vecteur de poids θ minimisant la fonction d'erreur choisie serait de déplacer au hasard θ dans l'espace de valeurs qu'il peut prendre, c'est-à-dire d'en faire varier légèrement le contenu, puis d'évaluer le modèle avec $E(\theta)$ pour voir si l'erreur a baissé.

Une façon plus structurée et plus efficace serait de déplacer θ dans l'espace de paramètres dans la direction réduisant le plus la fonction d'erreur. Cette direction est indiquée par le gradient inverse de la fonction d'erreur, soit $\nabla E(\theta)$. L'objectif devient

donc de trouver la valeur de θ telle que le gradient se dissipe, par l'équation :

$$\nabla E(\theta) = 0. \tag{1.11}$$

Autrement, $\nabla E(\theta) > 0$ indique qu'il est possible de faire un pas dans la direction de $-\nabla E(\theta)$ et donc de réduire encore plus l'erreur. Nous obtenons donc l'algorithme de *descente de gradient* [12], défini par :

$$\theta := \theta - \eta \nabla E(\theta). \tag{1.12}$$

Par contre, effectuer une mise à jour des poids à partir de tout l'ensemble d'entraînement apporte son lot de problèmes. En pratique, il est préférable d'effectuer une *descente de gradient stochastique* [12]

$$\theta := \theta - \eta \nabla E_n(\theta), \tag{1.13}$$

où $E_n(\theta)$ est l'erreur sur une seule donnée ou un groupe de données, appelé communément une *mini-batch*.

1.3 Rétropropagation

Maintenant que la fonction d'erreur est définie, nous devons établir comment propager le signal d'erreur afin d'apporter les modifications nécessaires aux poids du réseau. Pour ce faire, nous nous tournerons vers l'algorithme de *rétropropagation* [112] :

L'algorithme de rétropropagation, tel qu'illustré à la figure 1.3, utilise la règle de dérivation en chaîne afin de calculer le gradient de la fonction d'erreur par rapport à chaque activation : puisque l'activation de chaque couche dépend de la sortie de la couche précédente, le gradient d'erreur pour chaque couche doit être calculé par rapport au gradient de la couche subséquente. Plus spécifiquement, et pour notre exemple de réseau à quatre couches présenté précédemment, le gradient de l'erreur par rapport à l'activation a_l dans la couche de sortie (3) (dans le cas de la régression)

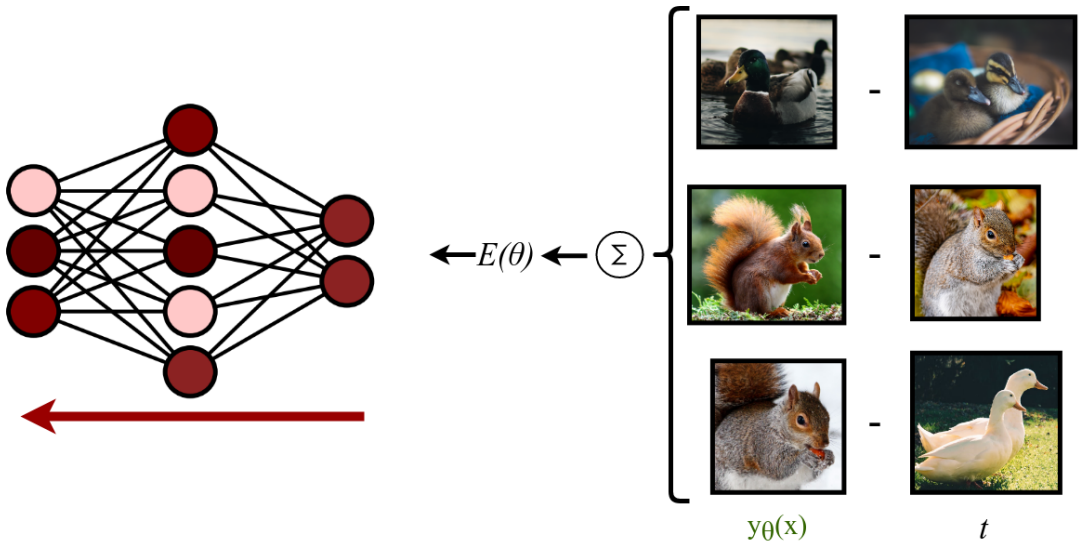

Figure 1.3 – Illustration de la propagation arrière de l'information, où les tons de rouge représentent le gradient appliqué à chaque neurone.

sera défini en tant que :

$$\frac{\partial E_n}{\partial a_l} = \delta_l = y_l - t_l. \tag{1.14}$$

Afin d'obtenir le gradient de l'erreur par rapport aux poids de la couche de sortie (3), nous devons utiliser la règle de dérivation en chaîne :

$$\frac{\partial E_n}{\partial w_{lk}} = \frac{\partial E_n}{\partial a_l}\frac{\partial a_l}{\partial w_{lk}} = \delta_l h_k. \tag{1.15}$$

Ensuite, nous pouvons *rétropropager* le signal d'erreur à l'activation de la couche cachée (2) selon :

$$\frac{\partial E_n}{\partial a_k} = \frac{\partial E_n}{\partial a_l}\frac{\partial a_l}{\partial a_k} = \delta_k = h'(a_k) w_{kj} \delta_l, \tag{1.16}$$

pour obtenir la dérivée par rapport aux poids de la couche cachée (2) :

$$\frac{\partial E_n}{\partial w_{kj}} = \frac{\partial E_n}{\partial a_l}\frac{\partial a_l}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}} = \delta_k h_j. \tag{1.17}$$

De façon plus générale, nous pouvons obtenir la dérivée d'une couche cachée par

rapport à la couche suivante selon :

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k, \tag{1.18}$$

où k sont les poids de la couche précédente.

Nous pouvons donc observer que l'information se propage de l'avant vers l'arrière du réseau de façon récursive afin de faire une prédiction, puis se *rétropropage* de l'arrière vers l'avant toujours de façon récursive afin d'ajuster les poids proportionnellement à leur contribution à l'erreur de prédiction calculée en sortie. La propagation telle que définie devrait être facilement extensible jusqu'à la couche d'entrée (1).

1.4 Conclusion

Dans ce chapitre, nous avons abordé le contexte de l'apprentissage supervisé et les réseaux de neurones. Nous avons vu comment les définir, comment définir une fonction de perte ainsi que son utilisation afin d'*entraîner* le réseau pour qu'il produise l'information souhaitée.

Alors que les réseaux de neurones sont un outil très puissant et que l'apprentissage supervisé a donné des résultats des plus extraordinaires ces dernières années, il n'est pas toujours possible d'obtenir une cible t, une vérité terrain, pour les données que nous avons. Dans le prochain chapitre, nous aborderons une alternative à l'apprentissage supervisé permettant d'apprendre par essais-erreur, sans avoir de cible prédéfinie.

"*As human agents, we are accustomed to operating with rewards that are so sparse that we only experience them once or twice in a lifetime, if at all.*"

—Pathak et. al, Curiosity-driven Exploration by Self-supervised Prediction

I pity the author

—LazyOptimist, reddit.com

Chapitre 2

Apprentissage par renforcement

L'apprentissage par renforcement a connu un engouement incroyable ces dernières années grâce à des avancées dans le domaine des jeux sur table (AlphaGo [126]), des jeux vidéos (Atari, AlphaStar [95, 145]), le contrôle robotique [129] et bien plus. L'apprentissage par renforcement existe depuis bien des années : les termes *reinforcement* (renforcement) et *reinforcement learning* (apprentissage par renforcement) utilisés dans le contexte qui nous intéresse firent leur apparition dans les années 1960 [94].

Par contre, le premier succès grand publique attendra à 1992 avec la sortie de TD-Gammon [138] : un programme utilisant un réseau de neurones jouant au jeu de Backgammon, entraîné en jouant contre lui-même (par *self-play*), permettant d'avoir des performances égales à celles des meilleurs joueurs de Backgammon.

Dans ce chapitre, nous aborderons le problème de l'apprentissage par renforcement. Nous le formaliserons grâce aux Processus de Décisions Markoviens (PDM), puis nous ferons un bond vers les réseaux de neurones et offriront une comparaison avec une autre méthode d'apprentissage : l'apprentissage supervisé. Finalement, nous explorerons l'apprentissage par renforcement profond.

2.1 Apprentissage par renforcement

La méthode essai-erreur est une forme d'apprentissage très intuitive pour l'être humain. Du plus jeune âge jusqu'à sa mort, l'humain interagit avec son environnement afin d'en savoir plus sur celui-ci et lui-même. Que ce soit pour apprendre à marcher ou cuisiner, conduire ou programmer, l'être humain ne dispose souvent pas de démonstrations explicites de ce qu'il doit faire pour accomplir ses buts. Il doit, en faisant des tentatives, en effectuant des actions ayant des conséquences incertaines, apprendre le résultat de ses manipulations. Puis, suite à de nombreuses chutes et recettes brûlées, l'être humain apprend à poser les actions le menant à ses buts.

L'apprentissage par renforcement consiste à apprendre comment agir, comment sélectionner l'action appropriée selon la situation, afin de maximiser une fonction appelée la *récompense*. Cette récompense, une fois maximisée, signifie généralement que le but sous-jacent est atteint.

Prenons l'exemple d'un bras robotique tentant d'apprendre à empiler des blocs sur une table. Nous nommerons l'entité apprenante par renforcement l'«agent», et celui-ci agira selon une marche à suivre qu'il tentera d'améliorer, nommée la politique. Dans ce cas-ci, l'environnement de l'agent serait la table sur laquelle repose les blocs, les blocs ainsi que le bras lui-même. Les actions que l'agent peut effectuer seraient le couple à appliquer aux joints du bras robotique afin de modifier sa configuration, et la récompense pourrait être la hauteur du bloc le plus élevé. Le processus d'apprentissage serait par exemple de disposer les blocs de façon aléatoire sur la table, de laisser le robot bouger, expérimenter, lui donnant un signal de récompense à chaque action qu'il effectue, puis, à la fin de l'épisode d'entraînement, par exemple après quelques minutes, arrêter le robot, replacer les blocs aléatoirement sur la table, puis recommencer le processus.

Par contre, il demeure à ce jour très difficile d'entraîner des agents dans le monde réel [31, 164]. De ce fait, la plupart des percées sont faites par des agents virtuels agissant dans des simulateurs [137, 8]. Ces environnements permettent un apprentissage grandement accéléré, car le processus d'entraînement n'est plus contraint par des interactions humaines pour recommencer. Un autre exemple d'agent virtuel serait un agent jouant au jeu *Breakout* dans l'*Atari Learning Environment* [15]. L'agent serait

Figure 2.1 – Breakout, pour le Atari 2600. L'encadré rose représente l'état du jeu, soit les pixels affichés à l'écran. L'encadré vert entoure le score, adaptable en une récompense utilisable pour l'agent. L'encadré jaune représente la palette, que l'agent peut déplacer à gauche ou à droite.

représenté par le joueur du jeu, l'environnement serait le jeu lui-même et les actions seraient de bouger la raquette à gauche ou à droite dans le but de projeter la balle vers les briques en haut de l'écran. Dans ce cas-ci, la récompense est très explicite : le score du jeu. Les épisodes seraient représentés par les différentes parties que l'agent jouerait.

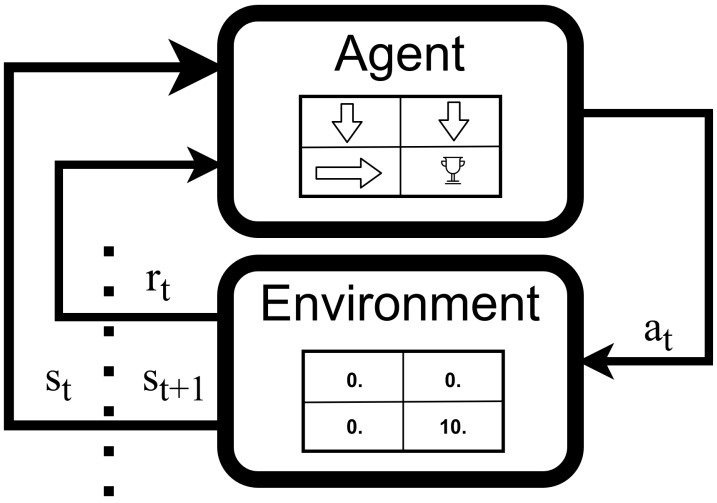

Figure 2.2 – Exemple de la "boucle" d'apprentissage par renforcement, où la politique et l'environnement prennent une forme tabulaire.

2.1.1 Processus de Décision Markoviens

Maintenant que le contexte est établi, nous pouvons formaliser les notions de politique, d'agent, d'environnement et de récompense.

Tel que mentionné, l'entité apprenante est nommée agent. L'agent interagira avec son environnement, qui englobe tout ce qui n'est pas l'agent. L'agent évaluera l'état de l'environnement, posera une action en conséquence, et l'environnement répondra en présentant un nouvel état à l'agent ainsi que le signal de récompense associé à cette action.

Les processus de décisions markoviens, une extension des chaînes de Markov [63] et représentés par le tuple (S, A, Pr, R), présentent un modèle permettant d'opérer sur ces concepts. À chaque temps $t = 0, 1, 2, ..., T$ l'agent reçoit l'état $s_t \in S$ de l'environnement et choisi une action $a_t \in A$ selon sa politique $\pi(a|s)$ [1]. L'état de l'environnement s'en retrouve alors modifié, et l'état réagit en renvoyant son nouvel état s_{t+1} ainsi que la récompense $r_t \in R \subset \mathbb{R}$ [2]. La figure 2.2 permet de visualiser ce

1. La notation $\pi(a|s)$ servira à dénoter la probabilité de sélection l'action a à l'état s selon la politique π, alors que la notation $\pi(s)$ dénotera la sélection de l'action a t.q. $a \leftarrow \pi(s)$.

2. La notation r_{t+1} est parfois employée pour désigner la récompense obtenue suite à l'action a_t en réaction à l'état s_t. Il serait logique de de faire de même, car après tout elle est reçue en même temps que l'état s_{t+1}. Par contre, la notation r_t sera néanmoins utilisée puisqu'il s'agit de la notation

processus.

Souvent, la transition de l'état s_t vers l'état s_{t+1} n'est pas déterministe. Par exemple, dans un contexte où un drône tenterait d'apprendre à voler, l'environnement pourrait être venteux. Le drône, suite à une action posée dans son environnement, pourrait donc se retrouver à un endroit différent selon la force du vent. Nous noterons les *dynamiques* de l'environnement par

$$p(s'|s,a) = Pr(S_{t+1} = s'|S_t = s, A_t = a), \tag{2.1}$$

soit la probabilité de se retrouver à l'état s' en étant précédemment à l'état s et en posant l'action a tel que $s, s' \in S,\ a \in A$. Il est à noter que la propriété définissant la probabilité d'atteindre un état futur ne dépend que de l'état courant (et l'action choisie) se nomme la propriété de Markov. Un processus, tel un Processus de Décision Markovien formalisé dans cette section, est dit markovien s'il respecte cette propriété. Plus formellement, la propriété de Markov est respectée si l'équation suivante est respectée :

$$p(s_{t+1}|s_0, a_0, s_1, a_1, ..., s_t, a_t) = p(s_{t+1}|s_t, a_t). \tag{2.2}$$

Finalement, nous appellerons *trajectoire* la suite de tuples d'états, actions et récompenses obtenues tout au long d'un épisode et la noterons par

$$\tau = (s_t, a_t, r_t), (s_{t+1}, a_{t+1}, r_{t+1}), (s_{t+2}, a_{t+2}, r_{t+2})..., \tag{2.3}$$

avec $|\tau| \leq |episode|$. Chaque épisode se termine par un état spécial appelé état *terminal* s_T, qui sera suivi d'une réinitialisation de l'environnement vers un état *initial* s_0, permettant le début d'un nouvel épisode.

Processus de décision markoviens partiellement observables

Il est commun de croiser dans la littérature la notation $o \in O$ pour représenter les *observations* retournées par l'environnement à l'agent, plutôt que l'état s. L'observation dans ce cas-ci est une sous-représentation, une version simplifiée de l'état. Pour notre exemple du jeu *Breakout*, l'observation o_t serait les pixels affichés à l'écran au temps t,

la plus couramment utilisée.

alors que l'état de l'environnement serait la valeurs de tous les registres composant la mémoire interne de la console virtuelle. Dans notre exemple de bras robotique, les observations seraient la position des différents joints articulés du bras, ainsi que la position des blocs. L'état de l'environnement serait plutôt la position de tous les objets dans la pièce ainsi que la définition de toutes les forces pouvant impacter le bras robotique et les blocs.

Afin de gérer ces situations, il est possible d'utiliser les Processus de Décision Markoviens partiellement observables (PDMPO), représentés par le tuple (S, A, Pr, R, Ω, O). Plutôt qu'un état $s \in S$, l'agent reçoit de l'environnement une observation $o \in \Omega$, une représentation de l'état réel de l'environnement. La fonction d'observation $\omega(s|o)$ note la probabilité d'être dans un état s en ayant l'observation o.

Afin de simplifier la notation et la représentation des environnements, il est courant d'utiliser la notation s pour représenter les observations, et implicitement de définir l'état de l'environnement comme étant l'état *observable* par l'agent en présupposant que l'observation retournée par l'environnement est suffisante pour avoir une bonne représentation de l'état.

Processus de décisions markoviens d'ordre supérieur

Le contexte de l'apprentissage par renforcement présume habituellement la présence d'un processus de décision markvovien d'ordre 1, signifiant que la transition vers l'état s_{t+1} ne dépende que de l'état s_t et de l'action a_t. Par contre, ce n'est pas toujours le cas. Par exemple, dans le jeu *Breakout*, la position de la balle au temps $t + 1$ dépend non-seulement de sa position au temps t, mais aussi de la vélocité de celle-ci, accumulée au cours de plusieurs pas de temps avant. Dans ce cas, la propriété de Markov n'est pas respectée et l'environnement est dit *non-markovien*.

Plusieurs pistes de solutions sont possibles afin de mitiger les effets ce problème, mais une solution courante consiste à envoyer à l'agent les n états précédents en plus de l'état courant afin d'ajouter une notion d'historique à l'agent [95].

2.1.2 Équation de Bellman

Jusqu'à présent, nous avons établi que l'agent tente d'accumuler le plus de récompense possible, ce qui signifierait que son but est atteint. Permettons nous de définir plus en détail ce que nous entendons par cela : nous tentons, à travers l'agent, de maximiser le *retour*, soit le cumul de récompenses obtenues, noté selon :

$$g_t = r_t + r_{t+1} + r_{t+2} + ...r_T, \tag{2.4}$$

où g_t représente la suite de récompenses à partir de l'instant courant jusqu'à la fin de l'épisode. Maximiser g revient donc à maximiser la récompense obtenue. Par contre, cette représentation pose problème : qu'arrive-t'il lorsque $t \to \infty$? Dans un contexte simplifié où l'agent reçoit une récompense $+1$ à chaque temps t, le retour tendrait vers l'infini. Rappelons que nous sommes dans un contexte informatique, où les valeurs infinies ne sont pas souhaitables. Nous utiliserons donc généralement comme facteur de performance le *retour escompté*, où

$$g_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \tag{2.5}$$

avec γ appelé *l'escompte*, $0 \leq \gamma < 1$. Cette notation [3] empêche g de tendre vers l'infini puisque $t \to \infty \implies \gamma_t \to 0$. Cet escompte a aussi la propriété intéressante, lorsque ajusté, de rendre l'agent plus ou moins vorace. En effet, un γ s'approchant de 1 rend les récompenses futures plus attrayantes, alors qu'un γ de 0 ne rend l'agent qu'attiré que vers les récompenses immédiates. Nous pouvons aussi au passage apprécier la qualité récursive de g. En effet :

$$\begin{aligned} g_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ... \\ &= r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3}) + ... \\ &= r_t + \gamma g_{t+1}. \end{aligned} \tag{2.6}$$

Rappelons que la récompense obtenue au temps t dépend de l'état s_t ainsi que

3. Puisque γ est un paramètre, il serait plus juste de noter le retour escompté par $g_t(\gamma)$. Par contre, l'usage de l'escompte étant la norme, le paramètre est presque toujours omis de la notation.

l'action a_t choisie par l'agent. Il est donc naturel de présumer que certaines actions mèneront vers des états plus rentables que d'autres. Par exemple, dans le cas d'une voiture autonome apprenant à conduire, un état respectant les limites de vitesse et indiquant que la voiture se trouve entre les lignes de la route est préférable à un état représentant la voiture fonçant à toute vitesse vers le rebord d'une falaise. Les actions choisies par la voiture dépendront de sa politique $\pi(a|s)$, la probabilité de poser l'action a à l'état s. La politique influencera en retour dans quelle séquence d'états la voiture se retrouvera aux temps futurs.

Nous pouvons voir que les récompenses futures dépendent donc de la politique de l'agent. De ce fait, dans le but d'évaluer la désirabilité, la *valeur*, d'un état, il est nécessaire de définir la fonction *état-valeur* (*state-value function*, ou *value function*) d'un état selon une politique :

$$v_\pi(s) = \mathbb{E}_\pi[g_t|s_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{T-k} \gamma^k r_{t+k}|s_t = s] \; \forall s \in S, \tag{2.7}$$

où $v_\pi(s)$ peut être interprété comme le retour espéré à l'état s en agissant selon π jusqu'à la fin de l'épisode. Il peut être intéressant de définir aussi la qualité d'une action posée sur un état. Nous définirons la fonction *action-valeur* (*action-value function*, ou *q-function*) comme étant :

$$q_\pi(s, a) = \mathbb{E}_\pi[g_t|s_t = s, a_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{T-k} \gamma^k r_{t+k}|s_t = s, a_t = a] \; \forall s \in S, a \in A. \tag{2.8}$$

Il est important de se rappeler de la propriété récursive de g. Nous pouvons donc définir $q_\pi(s, a)$ tel que :

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[g_t|s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_t + \gamma g_{t+1}|s_t = s, a_t = a] \\ &= \sum_{s'} p(s_{+1}|s_t, a_t)[r_t + \gamma \mathbb{E}_\pi[g_{t+1}]] \\ &= \sum_{s'} p(s_{+1}|s_t, a_t)[r_t + \gamma v_\pi(s_{t+1})]. \end{aligned} \tag{2.9}$$

L'équation 2.9 se nomme l'équation de Bellman [118]. Avec la dernière ligne de

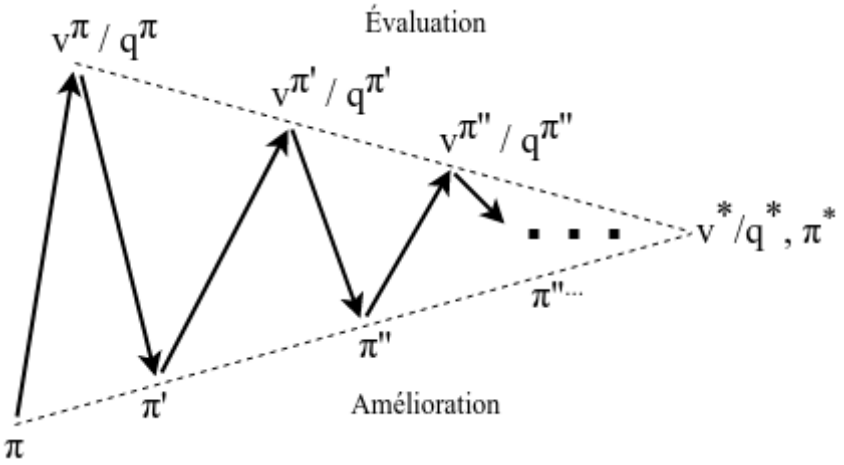

Figure 2.3 – Processus généralisé d'itération de politiques utilisé par les algorithmes d'apprentissage par renforcement. Le processus alterne entre deux étapes : l'amélioration progressive de la politique grâce à la fonction état-valeur/action-valeur et l'évaluation de la politique par la fonction état-valeur/action-valeur correspondante. Le but de l'apprentissage par renforcement est de trouver la paire de fonction état-valeur/action-valeur et politique optimale.

l'équation 2.9, on peut remarquer qu'il est possible de définir la fonction *action-valeur* en utilisant la fonction *état-valeur*. Inversement, il est possible de définir la fonction *état-valeur* avec la fonction *action-valeur* selon :

$$v_\pi(s) = \mathbb{E}[q_\pi(s, a)]. \tag{2.10}$$

Tel qu'illustré par la figure 2.3, l'apprentissage par renforcement consiste à trouver la meilleure paire de politique et fonction état-valeur/action-valeur, c'est à dire la politique π pour laquelle $v_\pi(s) \geq v_{\pi'}(s)\ \forall s \in S$. Nous utiliserons $*$ pour représenter l'optimalité d'une fonction. Par exemple, la politique optimale associée à une fonction *état-valeur* optimale sera notée π_*. Inversement, la fonction *action-valeur* optimale sera notée par :

$$q_*(s, a) = \sum_{s'} p(s'|s, a)[r + \gamma \max_{a'} q_*(s', a')]. \tag{2.11}$$

L'équation 2.11 se nomme l'équation d'optimalité de Bellman. Grâce à celle-ci, obtenir la politique optimale devient chose facile : la politique n'a qu'à regarder,

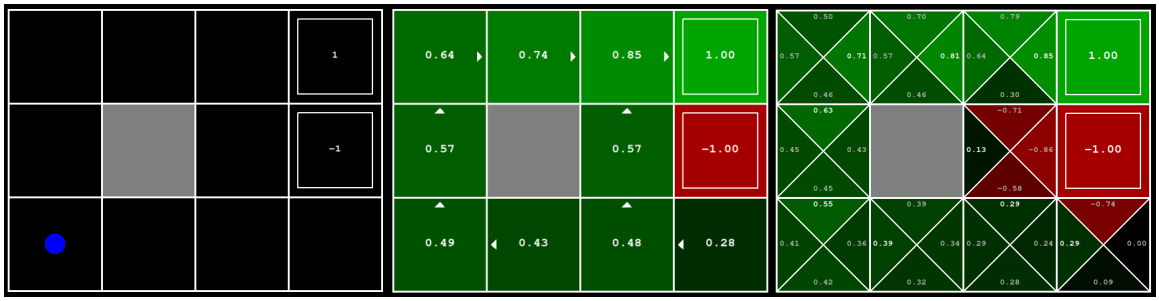

Figure 2.4 – Exemple d'environnement "jouet", ainsi que de fonctions état-valeur et action-valeur apprises. L'environnement possède un bruit de 0.2 appliqué aux actions, signifiant que les actions effectuées par l'agent ont 20% de chance de l'amener vers un autre état que celui visé. En bleu : la position initiale de l'agent. En gris : Une case inaccessible. À gauche : L'environnement, contenant un état terminal avec une récompense positive (1) et un état terminal avec une récompense négative (-1). Centre : Exemple de fonction état-valeur apprise, ainsi que d'une politique également apprise, représentée par la flèche en chaque case de l'environnement. Droite : Fonction action-valeur apprise, où la valeur de chaque paire (s, a) est indiquée dans le quartier de case correspondant.

pour chaque action possible à l'état s, celle qui maximise $q_*(s, a)$ et la choisir. Nous nommerons une politique agissant de cette façon (ne sélectionnant que les actions optimales) une politique *vorace* (*greedy*). La figure 2.4 offre un exemple d'environnement ainsi que de fonctions état-valeur et action-valeur apprises.

Malheureusement, obtenir une solution exacte à l'équation de Bellman s'avère, mis à part dans le cas d'exemples jouets, généralement impossible. Tenter de le faire reviendrait à faire une recherche exhaustive de toutes les paires (s, a) possibles et calculer la probabilité et la récompense associée. De plus, une solution à cette équation demande d'avoir accès aux vraies dynamiques de l'environnement, ce qui est rarement le cas. Finalement, l'équation d'optimalité de Bellman requiert que la propriété de Markov soit respectée, ce qui n'est aussi pas toujours le cas. Beaucoup de méthodes d'apprentissage par renforcement tenterons d'apprendre une solution approximative à 2.11 en échantillonnant des transitions dans l'environnement, et d'autre ne s'en préoccuperons même pas.

2.1.3 Politiques apprises par renforcement

Qu'arrive-t'il une fois le processus d'apprentissage (c.f. sections 2.2 et 2.3) terminé ? La politique apprise est alors jugée optimale et les composantes et paramètres de celle-ci sont fixés. La politique apprise peut donc, idéalement, être déployée dans le monde réel et appliquée au problème pour lequel elle a été entraînée. Elle peut maintenant remporter des tournois du jeu de table Go [126], gagner des parties du jeu vidéo Starcraft [145] ou conduire des voitures de façon autonome [76] . Il est important de noter que, tout comme les étiquettes ne sont plus nécessaires une fois le modèle entraîné dans le contexte de l'apprentissage supervisé (c.f. chapitre 1), la fonction de récompense n'est plus nécessaire une fois que la politique est entraînée. Après tout, la politique est jugée optimale et n'a plus rien à "apprendre" de la fonction de récompense. La politique ne se fiera alors entièrement qu'à la sortie de sa fonction état-valeur ou action-valeur correspondante (ou purement à soi-même, c.f. section 2.3.3) afin de déterminer l'action à poser selon l'état de l'environnement. Autrement, le processus est très similaire à celui illustré par la figure 2.2 : l'environnement envois un état initial à l'agent entraîné, l'agent produit une action, celle-ci est envoyée à l'environnement et l'environnement renvois le prochain état. Le processus se termine lorsqu'un état terminal est rencontré ; l'état terminal étant idéalement plus souhaitable que ceux rencontrés lors de l'entraînement.

2.2 Méthodes d'apprentissage classiques

Dans cette section, nous aborderons des algorithmes permettant de trouver une solution approximative à 2.11. Bien que très puissantes, ces méthodes sont limitées à des environnements ayant un espace d'états et d'actions assez restreint. Malgré tout, ces algorithmes introduisent des concepts qui sont encore pertinent à ce jour, et que nous retrouverons dans la section 2.3

2.2.1 Programmation dynamique

Jusqu'à présent, nous avons établi l'objectif à atteindre, mais nous n'avons pas établi comment s'y rendre. Remédions à ce problème. Plusieurs algorithmes par

programmation dynamique existent [40] et permettent, dans un contexte où les états sont énumérables et où les dynamiques de l'environnement sont connues, de calculer la fonction état-valeur optimale et donc d'obtenir la politique optimale. Des exemples de ces algorithmes incluent *Itération par politique* [11] et *Itération par valeur* [10].

Les deux algorithmes fonctionnent sous un principe similaire : à la première phase de l'algorithme, la valeur de chaque état est mise à jour selon la probabilité qu'il se produise, ainsi que la valeur de tous les états successeurs à celui-ci. La deuxième phase de l'algorithme consiste à mettre à jour la politique pour refléter les changements à la fonction d'évaluation d'états. Ce processus en deux phase est un concept très fréquent chez les algorithmes d'apprentissage par renforcement, et nous le nommerons *Itération par politique généralisé*. L'algorithme d'*itération par politique*, fonctionnant sous ce principe, est disponible dans l'encadré 2.1.

À noter que ces algorithmes ne calculent pas directement une solution à l'équation 2.9, mais en calculent plutôt itérativement une approximation convergeant vers la solution ; nous noterons ces estimés $Q(s,a), V(s) \approx q(s,a), v(s)$. De plus, en regardant les algorithmes, nous pouvons voir que ceux-ci font des estimés de valeur d'état basés sur d'autres estimés de valeur d'état. Il s'agit d'une technique utilisée très souvent, que nous nommerons *autoamorçage* (*bootstrap*).

Ces algorithmes requièrent plusieurs balayages complets de l'espace d'états et sont donc souvent intraitables hors d'environnements tabulaires. De plus, afin d'estimer correctement la probabilité qu'un état se produise, les dynamiques de l'environnement doivent être connues, ce qui n'est pas souvent possible.

2.2.2 Différence temporelle

Tel que mentionné dans la dernière section, il n'est pas souvent possible d'avoir accès à un modèle parfait des dynamiques de l'environnement, ou de l'énumération de tous les états possible. Bien que cela puisse sembler contre-intuitif, il est souvent plus facile d'échantillonner des transitions plutôt que d'avoir accès à la distribution explicite de celles-ci. Il serait donc intéressant de pouvoir interagir avec l'environnement, générer des transitions, possiblement des épisodes complets, puis d'apprendre des actions posées.

Algorithm 2.1 Itération par politique

1: $V(s) \in \mathbb{R}\ \forall s \in S$
2: $\pi(s) \in A(s)\ \forall s \in S$
3: **procedure** Évaluation de politique
4: **loop**
5: $\Delta \leftarrow 0$
6: **for all** $s \in S$ **do**
7: $v \leftarrow V(s)$
8: $V(s) \leftarrow \sum_{s'} p(s'|s, \pi(s))[r + \gamma V(s')]$
9: $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
10: **jusqu'à** $\Delta < \theta$ (un petit nombre)
11: **procedure** Amélioration de politique
12: $stable \leftarrow vrai$
13: **for all** $s \in S$ **do**
14: $action \leftarrow \pi(s)$
15: $\pi(s) \leftarrow \max_a \sum_{s'} p(s'|s, a)[r + \gamma V(s')]$
16: Si $action \neq \pi(s)$ alors $stable \leftarrow faux$
17: Si *stable* alors **retour** $V \approx v_*, \pi \approx \pi_*$, sinon aller à 3

Les méthodes à différence temporelle, comme les méthodes *Monte-Carlo* [118] (nous y reviendrons), permettent d'apprendre directement en échantillonnant l'environnement et en s'entraînant sur l'expérience récoltée. Les méthodes à différence temporelle sont toutes basées sur l'autoamorçage, introduit dans la dernière section, où une forme de la mise-à-jour suivante :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[\underbrace{r_t + \gamma Q(s_{t+1}, a_{t+1})}_{\text{cible}} - \underbrace{Q(s_t, a_t)}_{\text{prédiction}}] \tag{2.12}$$

est utilisée. L'objectif de ces méthodes, à travers l'échantillonnage de l'environnement et l'application de 2.12, est de minimiser la différence entre la cible et la prédiction.

Jusqu'à présent, nous avons exploré des politiques voraces, où l'action choisie est celle maximisant la sortie de la fonction *action-valeur*. Dans le cas des algorithmes par programmation dynamique, nous pouvons utiliser exclusivement ces politiques, car la nature tabulaire de l'algorithme permet d'explorer tous les états et d'en calculer la fonction *état-valeur*. Par contre, dans un contexte où un algorithme apprend par

essai-erreur, une politique ne sélectionnant que l'action jugée optimale au moment présent *exploiterait* toujours les mêmes états, sans *explorer*.

Ce dilemme entre l'exploitation et l'exploration d'états et d'actions est très présent en apprentissage par renforcement. En effet, aussi étrange que cela puisse paraître, il faut souvent avoir recourt à une politique sous-optimale afin de trouver la politique optimale. Une façon de procéder consiste à parfois sélectionner une action jugée optimale, appelée action *vorace*, et parfois sélectionner une action au hasard. Cette stratégie d'exploration se nomme ϵ-vorace et est définie par : [118]

$$a_t = \begin{cases} \max_{a_t} Q(s_t, a_t), & \text{avec probabilité 1-}\epsilon. \\ \text{hasard parmis } A(s), & \text{avec probabilité } \epsilon. \end{cases} \tag{2.13}$$

Avec les équations 2.12 et 2.13, ainsi que le cadre général des méthodes à différences temporelles mentionnées plus haut, il devient assez facile de concevoir un algorithme permettant de trouver la politique optimale. Un de ceux-ci se nomme Sarsa [113], en référence au tuple $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$.

Algorithm 2.2 Sarsa

$Q(s,a) \in \mathbb{R}\ \forall s \in S\ a \in A(s)$
$Q(s,\cdot) \leftarrow 0\ \forall s$ terminaux
procedure Sarsa($\alpha \in]0,1]$, $\epsilon > 0$)
    loop
        Initialiser s
        $a \leftarrow \pi(s)$ de façon ϵ-vorace
        while s n'est pas terminal **do**
            $r, s' \leftarrow env(a)$
            $a' \leftarrow \pi(s')$ de façon ϵ-vorace
            $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
            $s, a \leftarrow s', a'$

Quelques années avant, Watkins [150] présentait dans sa thèse de doctorat l'algorithme *Q-learning*, ou Apprentissage-Q. Cet algorithme a la particularité d'être *hors-politique*, par opposition aux algorithmes présentés jusqu'à présent, qui sont *en-politique*. Les algorithmes *hors-politique* permettent d'utiliser une politique différente pour récolter de l'expérience, la politique *comportementale* β, que celle qui est

entraînée, la politique *cible* π. Cela donne un avantage considérable par rapport aux algorithmes *en-politique*, qui doivent approximer une fonction-Q non pas par rapport à la politique optimale, mais bien une politique sous-optimale permettant l'exploration.

Les algorithmes en-politique ont été inventées avant et sont généralement plus simples à implémenter et comprendre. Les algorithmes hors-politiques ont par-contre le grand avantage de pouvoir apprendre de politiques qui ne sont potentiellement même pas « apprenantes », telles qu'un contrôleur symbolique ou une banque de données récoltée par un humain. Par contre, ceux-ci ont tendance à souffrir d'une plus grande variance et sont plus lents à converger [118].

Algorithm 2.3 Apprentissage-Q

$Q(s,a) \in \mathbb{R}\ \forall s \in S\ a \in A(s)$
$Q(s,\cdot) \leftarrow 0\ \forall s$ terminaux
procedure Q-Learning($\alpha \in]0,1]$, $\epsilon > 0$)
    loop
        Initialiser s
        while s n'est pas terminal **do**
            $a \leftarrow \beta(s)$ de façon ϵ-vorace
            $r, s' \leftarrow env(a)$
            $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$
            $s, a \leftarrow s', a'$

2.3 Apprentissage par renforcement profond

Tel que mentionné dans les sections précédentes, les méthodes d'apprentissage classiques présentées jusqu'à maintenant sont limitées à des environnements plutôt simplistes, représentables sous une forme tabulaire. Prenons l'exemple d'un problème où l'espace d'état est une image couleur mesurant 256 pixels de haut et large, avec 3 canaux RVB ayant chacun 256 valeurs possibles. L'espace d'état est alors bien trop grand pour être représenté sous forme tabulaire et le processus d'apprentissage nécessiterait l'énumération de toutes les images possibles selon ces dimensions.

En plus de nécessiter une quantité phénoménale de mémoire pour représenter cet espace d'états, une fonction *action-valeur* tabulaire serait très inefficace : les méthodes

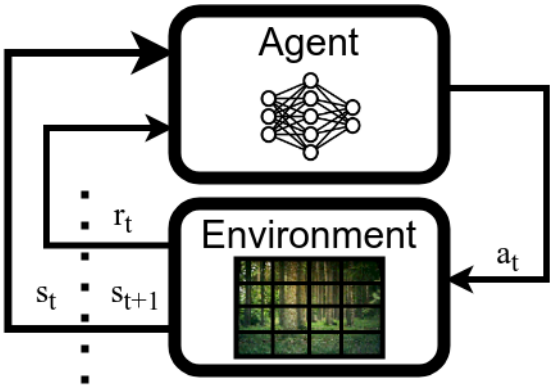

Figure 2.5 – Exemple de la "boucle" d'apprentissage par renforcement profond, où la politique est représentée par un réseau de neurone et l'environnement envoie comme état les pixels d'une caméra. Le principe demeure le même qu'avec l'apprentissage par renforcement "classique", mais le domaine d'états et la politiques peuvent être bien plus complexes.

de mise-à-jour de la fonction *action-valeur* ne mettent à jour la valeur que pour un seul état et action précis, alors qu'il est facile de s'imaginer que deux images très semblables auraient une valeur tout aussi similaire l'une à l'autre.

Il serait donc souhaitable d'avoir une fonction *état-valeur* ou *action-valeur* capable de *généraliser* sur plus d'un état, ainsi qu'une fonction représentable de façon plus compacte qu'une table comprenant une entrée pour tous les états et possiblement toutes les action. Pour ce faire, nous nous tournerons vers les *approximateurs de fonctions*, et plus spécifiquement les *réseaux de neurones* présentés au chapitre précédent. L'utilisation de réseaux de neurones profonds comme approximateurs de fonction dans l'apprentissage par renforcement donnera naissance au terme *apprentissage par renforcement profond*, et le processus est illustré à la figure 2.5.

Les méthodes d'apprentissage par renforcement profond ont l'avantage d'être à peine plus compliquées, mais grandement plus puissantes que les méthodes non-approximatives. Dans cette section, nous aborderons quelques algorithmes pertinents au sujet de ce mémoire, commençant avec une version modifiée de l'algorithme Sarsa présenté plus tôt, puis une version modifiée de l'Apprentissage-Q ayant eu des résultats étonnants sur la suite *Atari Learning Environment* [15]. Nous aborderons ensuite les méthodes de *gradient de politique* à travers l'algorithme *REINFORCE*, puis les

très populaires méthodes basées sur le concept d'*Acteur-Critique*, intégrant à la fois les méthodes de gradient de politique tout en maintenant des approximateurs de fonctions explicites. Nous aborderons l'algorithme d'*optimisation proximale de politique* (Proximal Policy Optimization, PPO) puis les méthodes de *gradient de politique déterministes* avec l'algorithme de *gradient de politique profond déterministe* (Deep Deterministic Policy Gradient, DDPG) et son amélioration directe : le *gradient de politique déterministe jumelé et différé* (Twin-Delayed Deep Deterministic Policy Gradient, TD3). Finalement, nous terminerons avec la présentation de l'algorithme *Soft acteur-critique* (Soft Actor-Critic, SAC).

2.3.1 Sarsa avec approximateurs de fonction

Permettons-nous d'introduire la notation $v_\theta(s)$ et $q_\theta(s,a)$ comme étant respectivement des fonctions *état-valeur* et *action-valeur* paramétrées par des réseaux de neurones (ainsi que les estimations respectives $Q_\theta(s,a)$ et $V_\theta(s)$), où le but demeure de trouver, par exemple, $q_\theta(s,a) \approx q_*(s,a)$. Alors que jusqu'à présent, nous travaillions avec des méthodes qui mettaient à jour la valeur d'une paire état-action $q(s,a)$ dans une table vers une *cible* u [4] (rappelons-nous l'équation 2.12, où $u = r_t + \gamma Q(s_{t+1}, a_{t+1})$) pour exactement cette action et valeur précisément, nous travaillerons maintenant avec des méthodes propageant la mise à jour à plusieurs actions et états, grâce à la puissance de généralisation des réseaux de neurones.

Avec une fonction *action-valeur* représentée par un réseau de neurones, et donc différentiable, nous souhaitons minimiser la fonction d'erreur rappelant fortement 1.9

$$J(\theta) = \frac{1}{2}\mathbb{E}_{s,a\sim\pi}\Big[u - q_\theta(s,a)\Big]^2 \tag{2.14}$$

où l'espérance viens du fait que les états rencontrés et les actions posées dépendent de la probabilité de les rencontrer selon la politique π. Nous pouvons donc nous servir du gradient de l'équation 2.14 afin de trouver les poids minimisant $J(\theta)$ grâce à la

4. Alors qu'en apprentissage supervisé, t est souvent utilisé pour dénoter la *cible*, u sera utilisé dans le cadre de l'apprentissage par renforcement puisque t était déjà pris pour dénoter les pas de temps.

Algorithm 2.4 Sarsa avec approximateurs de fonction

$\theta \leftarrow$ initialisés aux choix de l'utilisateur
procedure Sarsa($\eta,\ \epsilon > 0$)
    loop
        Initialiser s
        $a \leftarrow \pi(s)$ de façon ϵ-vorace
        while s n'est pas terminal **do**
            $r, s' \leftarrow env(a)$
            $a' \leftarrow \pi(s')$ de façon ϵ-vorace
            $U \leftarrow r_t + \gamma Q_\theta(s_{t+1}, a_{t+1})$
            $\theta \leftarrow \theta - \eta\Big[U - Q_\theta(s, a)\Big]\nabla Q_\theta(s, a)$
            $s, a \leftarrow s', a'$
        $\theta \leftarrow \theta - \eta\Big[r_t - Q_\theta(s, a)\Big]\nabla Q_\theta(s, a)$ ▷ // Pas d'états subséquents

descente de gradient stochastique présentée dans l'équation 1.13 :

$$\theta := \theta - \eta\nabla J_{s,a}(\theta). \tag{2.15}$$

Plus souvent qu'autrement, u sera aussi une cible approximative, dénotée par U. Cette cible pourrait être, par exemple, un estimé de u collecté en interagissant plusieurs fois avec l'environnement. Mis au long, nous obtenons la mise à jour générale des poids suivante :

$$\theta := \theta - \eta\Big[U - q_\theta(s, a)\Big]\nabla q_\theta(s, a) \tag{2.16}$$

Grâce à la fonction d'erreur et la mise à jour des poids présentées dans cette section, nous avons tout le nécessaire pour revoir l'algorithme Sarsa (c.f Algorithme 2.2) présenté précédemment. En effet, il est très facile d'adapter la mise à jour des poids à l'équation 2.16 afin d'utiliser l'autoamorçage pour se créer une cible U. La mise à jour des poids pour l'algorithme *Sarsa avec approximateurs de fonction* (Algorithme 2.4) devient donc

$$\theta := \theta - \eta\Big[r_t + \gamma Q_\theta(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t)\Big]\nabla Q_\theta(s_t, a_t). \tag{2.17}$$

2.3.2 DQN

Les jeux vidéos sont récemment devenus un domaine où les agents entraînés par apprentissage par renforcement profond excellent et vont même jusqu'à obtenir des performances sur-humaines. Le travail de Minh et al. [95] fut un des premiers à exposer la puissance des réseaux de neurones profonds, particulièrement les réseaux de neurones convolutifs, lorsque mis au service de l'apprentissage par renforcement.

Leur approche, qu'ils nommeront *réseau Q profond* (Deep Q-network, DQN) fut de modifier l'algorithme d'apprentissage Q (2.3) afin d'utiliser les réseaux de neurones convolutifs, en plus de réutiliser les transitions récoltées à travers l'*experience replay* [89].

L'*experience replay* permet à l'agent apprenant de réutiliser les transitions en interagissant avec l'environnement en stockant dans un tampon D les tuples (s_t, a_t, r_t, s_{t+1}) rencontrées à chaque pas de temps. Ce tampon accumulera des transitions de plusieurs parties du même jeu et, à chaque pas de temps, une *mini-batch* (S, A, R, S') en sera rééchantillonné uniformément et utilisée pour mettre à jour les poids afin de permettre au réseau d'apprendre de ses expériences courantes, mais aussi de ses expériences passées. Rappelons nous que l'algoritme d'apprentissage Q est *hors-politique*, lui permettant d'apprendre de (presque) m'importe quelle politique, incluant les politiques passées du même agent.

La deuxième amélioration viens de l'utilisation d'une version *antérieure* du même réseau Q pour effectuer l'autoamorçage. En effet, souvenons-nous que l'autoamorçage est une forme d'apprentissage supervisé où la cible est créée en partie avec une vérité terrain (la récompense r_t) ainsi que la propre estimation du réseau ($\gamma Q(s_{t+1}, a_{t+1})$). Si le même réseau est utilisé pour la prédiction et la cible, et que celui-ci est mis à jour à chaque pas de temps, le réseau tente effectivement de se rapprocher d'une "cible mouvante", puisque l'estimation de $Q(s_{t+1}, a_{t+1})$ change aussi à chaque pas de temps. La solution abordée fut donc de retarder la mise à jour du réseau "cible" qu'à chaque C pas de temps, ayant pour effet de "stabiliser" la cible vers laquelle le réseau apprenant tente de se rapprocher, et réduisant ainsi de beaucoup la variance lors de l'apprentissage.

Algorithm 2.5 DQN

$\theta \leftarrow$ initialisés aux choix de l'utilisateur
$\hat{\theta} \leftarrow \theta$
$D = \{\}$
procedure DQN(η, $\epsilon > 0$, $C > 0$)
    loop
        Initialiser s
        while s n'est pas terminal **do**
            $a \leftarrow \beta(s)$ de façon ϵ-vorace
            $r, s' \leftarrow env(a)$
            $D \leftarrow D \cup (s, a, r, s')$
            échantillonner de D une mini-batch de transitions (S, A, R, S')
            $U \leftarrow R + \gamma max_{A'} Q_{\hat{\theta}}(S', A')$
            $\theta \leftarrow \theta - \eta\Big[U - Q_\theta(S, A)\Big]\nabla Q_\theta(S, A)$
            $s \leftarrow s'$
            à tous les C : $\hat{\theta} \leftarrow \theta$

La mise à jour des poids du réseau prend alors la forme suivante :

$$\theta := \theta - \eta\Big[r + \gamma max_{a'} Q_{\hat{\theta}}(s', a') - Q_\theta(s, a)\Big]\nabla Q_\theta(s, a), \tag{2.18}$$

où $\hat{\theta}$ dénote les poids antérieurs du réseau.

L'algorithme fut entraîné sur la suite *Atari Learning Environment* [15](ALE), constituée de 57 jeux originellement sortis sur la console Atari 2600 et faisant maintenant partie de la culture populaire tels que *Pong* ou *Breakout*. Le simulateur de la suite renvoie les images "brutes" telles qu'elles seraient affichées à un écran et accepte en entrée les interactions possibles envers le contrôleur "virtuel" de la console simulée. Les auteurs ont choisi de définir la récompense comme étant $+1$ si le score du jeu augmente par rapport à l'image précédente, 0 s'il reste inchangé et -1 s'il baisse. Cette dernière distinction permet de standardiser la récompense parmis tous les jeux.

Afin de réduire les ressources nécessaires à l'entraînement de DQN, les auteurs ont réduit les images en entrée d'une résolution de 210x160 pixels avec 128 couleurs à une carte de "luminance" de 84x84 pixels. De plus, certains jeux requièrent plus d'informations que ce qui est affiché en tout temps à l'écran afin d'en avoir une bonne compréhension de celui-ci. Un bon exemple est le jeu *Pong*, où la position

est disponible qu'en ayant la dernière image du jeu. Par contre, pour bien estimer la vélocité de la balle et donc pouvoir placer la palette au bon endroit, plusieurs images successives du jeu sont nécessaires. On peut donc estimer que les observations renvoyées par le simulateur ne permettent pas de bien capturer l'état du jeu, rendant le PDM partiellement observable. Afin de palier à ce problème, les auteurs ont plutôt choisi d'envoyer au réseau les quatre dernières images produites par le simulateur.

Grâce aux améliorations décrites dans cette section, les auteurs avaient réussi à surpasser en performance tous les algorithmes précédemment proposés, ainsi que d'obtenir des performances semblables à celles d'un testeur de jeux vidéos professionnel humain sur 49 des 57 jeux.

2.3.3 Gradient de politique et Acteur-Critique

Introduits en 1992 [154], les méthodes par gradient de politique (*policy gradient*) tentent d'apprendre directement la politique sans passer par une fonction *action-valeur*. La politique devient directement paramétrée par θ, et nous noterons $\pi_\theta(a|s)$ la probabilité de choisir l'action a en sachant que l'environnement est dans l'état s selon les poids θ.

Nous obtiendrons la probabilité d'une action en demandant au réseau de prédire une *préférence* $h_\theta(s, a)$ pour chaque action et état. Plus la préférence est grande, plus souvent l'action est choisie, mais il n'y a pas de lien direct entre la récompense espérée ; seule la préférence d'une action relative aux autre actions est importante. L'action avec la plus grande préférence peut ensuite être choisie selon, par exemple, un softmax dans le cas d'actions discrètes.

Les méthodes par gradient de politique ont l'avantage d'être facilement extensible à des espaces d'actions continus. En effet, l'utilisation d'une fonction *action-valeur* pour déterminer l'action optimale dans un espace d'actions continu demande de vérifier une infinité de valeurs. Pour palier à ce problème, nous pouvons définir la politique comme une densité de probabilité sur l'action par une gaussienne :

$$\pi_\theta(a|s) = \frac{1}{\sigma_\theta(s)\sqrt{2\pi}} e^{\left(-\frac{(a-\mu_\theta(s))^2}{2\sigma_\theta(s)^2}\right)}, \tag{2.19}$$

où μ_θ et σ_θ sont les sorties d'un réseau de neurones.

Peu importe que le domaine d'action soit discret ou continu, nous devons nous définir une mesure de *performance* pour évaluer la politique et obtenir un gradient à propager. Une mesure intuitive serait d'utiliser la fonction *état-valeur* à partir de l'état initial s_0 tel que

$$J(\theta) = v_{\pi_\theta}(s_0), \tag{2.20}$$

que nous tenterons d'estimer à travers l'échantillonage de l'environnement pour ensuite effectuer une *ascension de gradient* (puisque nous voulons maximiser cette mesure de performance) telle que

$$\theta := \theta + \eta \nabla J(\theta). \tag{2.21}$$

Le théorème du gradient de politique [118] dicte que

$$\nabla J(\theta) \propto \sum_s p_\pi(s) \sum_a q_\pi(s,a) \nabla \pi_\theta(a|s). \tag{2.22}$$

où $p_\pi(s)$ est la probabilité de rencontrer l'état s en suivant π. Le théorème du gradient de politique donne une expression *exacte* proportionelle au gradient de la politique, mais nous pouvons approximer cette expression en échantillonnant l'environnement tel que l'échantillonnage est proportionnel à cette expression ; le quotient de proportionalité peut ensuite être absorbé dans le pas de gradient η. Puisque la partie droite de l'équation 2.22 est une somme sur les états, pondérée par la probabilité de rencontrer cet état, il est naturel de présumer que suivre la politique π devrait conserver cette probabilité et il en va de même pour l'action choisie.

Algorithm 2.6 REINFORCE

$\theta \leftarrow$ initialisés aux choix de l'utilisateur
procedure REINFORCE(η)
    loop
        Générer un épisode $(s_0, a_0, r_0), (s_1, a_1, r_1), ..., (s_t, a_t, r_t)$ en suivant $\pi_\theta(\cdot|\cdot)$
        for all t dans t = 0,1,...T **do**
            $g \leftarrow \sum_{k=t}^{T} \gamma^{k-t} r_k$
            $\theta \leftarrow \theta + \eta \gamma^t g \nabla \log \pi_\theta(a_t|s_t)$

Nous pouvons donc transformer l'équation 2.22 en

$$\begin{aligned}
\nabla J(\theta) &\propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi_\theta(a|s) \\
&= \mathbb{E}_{s\sim\pi}\Big[\sum_a q_\pi(s,a) \nabla \pi_\theta(a|s)\Big] \\
&= \mathbb{E}_{s\sim\pi}\Big[\sum_a \pi_\theta(a|s) q_\pi(s,a) \frac{\nabla \pi_\theta(a|s)}{\pi_\theta(a|s)}\Big] \\
&= \mathbb{E}_{s,a\sim\pi}\Big[q_\pi(s,a) \frac{\nabla \pi_\theta(a|s)}{\pi_\theta(a|s)}\Big] \\
&= \mathbb{E}_{s,a\sim\pi}\Big[g \frac{\nabla \pi_\theta(a|s)}{\pi_\theta(a|s)}\Big]. \qquad (2.23)
\end{aligned}$$

La dernière ligne nous permet d'effectuer une ascension de gradient stochastique sur les poids ayant le nom de REINFORCE [154], définie par :

$$\theta := \theta + \eta g_t \nabla \log \pi_\theta(a_t|s_t), \qquad (2.24)$$

où on peut noter que le ratio $\frac{\nabla \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)}$ a été remplacé par $\nabla \log \pi_\theta(a_t|s_t)$ en suivant l'identité $\nabla \log x = \frac{\nabla x}{x}$.

Cette mise à jour a plusieurs propriétés intéressantes : elle augmente la probabilité de répéter l'action a_t lorsque l'état s_t est rencontré de façon proportionnelle au retour associé à cette paire d'action/état, mais de façon inversement proportionnelle à la probabilité d'effectuer l'action. Ceci permet aux actions ayant un plus grand retour d'être plus probables, sans accorder trop d'importance aux actions déjà très probables pour éviter qu'elles ne prennent le dessus sur des actions moins probables, mais ayant

possiblement un plus grand retour. L'algorithme complet est disponible dans l'encadré 2.6.

On peut aussi noter que l'algorithme génère des épisodes complets afin de calculer le retour. Ce type d'algorithme se nomme *Monte-Carlo* [118] et possède quelques avantages sur les méthodes à différence temporelle abordées jusqu'à maintenant. Notamment, ces méthodes ne souffrent pas de l'instabilité des méthodes par différence temporelle puisqu'elles ne sont pas biaisées par leurs propres estimations ; leur objectif est basé sur la vérité terrain obtenue lors d'échantillonage de l'environnement. Par contre, les méthodes Monte-Carlo ont tendance à souffrir d'une plus grande variance puisque leur mise à jour n'est pas seulement basée sur le prochain tuple $(s_{t+1}, a_{t+1}, r_{t+1})$, mais sur la trajectoire complète. De plus, le recours à des épisodes complets peut aussi empêcher l'utilisation de *mini-batch* lors de la mise à jour si les épisodes sont de taille variable.

Gradient de politique avec référentiel et Acteur-Critiques

Une technique très utilisée afin de réduire la variance de l'algorithme REINFORCE est de comparer la sortie de la fonction *action-valeur* à un référentiel $b(s)$ tel que

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \Big(q_\pi(s,a) - b(s)\Big) \nabla \pi_\theta(a|s), \tag{2.25}$$

pour obtenir la mise à jour *REINFORCE avec référentiel* définie selon :

$$\theta := \theta + \eta\Big(g_t - b(s_t)\Big) \nabla \log \pi_\theta(a_t|s_t). \tag{2.26}$$

Le référentiel peut être n'importe quelle fonction (même une fonction constante), tant qu'il ne varie pas selon l'action a. Un candidat naturel pour incarner le référentiel est une fonction *état-valeur*, elle aussi définie par un réseau de neurones ayant les poids ϕ et apprenant grâce au retour accumulé de l'épisode. Utiliser une fonction *état-valeur* donne lieu au concept d'*avantage* :

$$\mathcal{A}(s_t, a_t) = q(s_t, a_t) - v(s_t), \tag{2.27}$$

permettant de favoriser les actions ayant un retour plus grand que la moyenne. Nous

Algorithm 2.7 REINFORCE avec référentiel

$\theta \leftarrow$ initialisés aux choix de l'utilisateur
$\phi \leftarrow$ initialisés aux choix de l'utilisateur
procedure REINFORCE avec référentiel(η_θ, η_ϕ)
    loop
        Générer un épisode $(s_0, a_0, r_0), (s_1, a_1, r_1), ..., (s_t, a_t, r_t)$ en suivant $\pi_\theta(\cdot|\cdot)$
        for all t dans t = 0,1,...T **do**
            $g_t \leftarrow \sum_{k=t}^{T} \gamma^{k-t} r_k$
            $\mathcal{A}_t \leftarrow g_t - V_\phi(s_t)$
            $\theta \leftarrow \theta + \eta_\theta \gamma^t \mathcal{A}_t \nabla \log \pi_\theta(a_t|s_t)$
            $\phi \leftarrow \phi - \eta_\phi \mathcal{A}_t \nabla V_\phi(s_t)$

obtenons la mise à jour des poids *avec avantage*

$$\theta := \theta + \eta\Big(g_t - v(s_t)\Big)\nabla \log \pi_\theta(a_t|s_t). \tag{2.28}$$

Cette formulation a la propriété de "formaliser" le concept d'*essai-erreur* : les actions donnant un résultat supérieur à celui espéré sont rendues plus probables, et les actions donnant un résultat inférieur sont rendues moins probables.

Dans le but de réduire la variance, une extension logique aux méthodes de politique de gradient avec référentiel Monte-Carlo est d'utiliser la fonction *état-valeur* apprise par un réseau de neurones afin de faire de l'autoamorçage, transformant l'algorithme Monte-Carlo (c.f Algorithme 2.7) en algorithme par différence temporelle (Algorithme 2.8). La mise à jour des poids pour cette nouvelle méthode, nommée *Acteur-Critique* devient donc :

$$\begin{aligned}\theta &:= \theta + \eta\Big(g_t - v_\phi(s_t)\Big)\nabla \log \pi_\theta(a_t|s_t) \\ &:= \theta + \eta\Big(r_t + \gamma v_\phi(s_{t+1}) - v_\phi(s_t)\Big)\nabla \log \pi_\theta(a_t|s_t) \\ &:= \theta + \eta \mathcal{A}_t \nabla \log \pi_\theta(a_t|s_t),\end{aligned} \tag{2.29}$$

en se rappelant que $q(s_t, a_t) = r_t + \gamma v(s_{t+1})$. Le pseudo-code pour une méthode *Acteur-Critique en-politique* est donné dans l'encadré 2.8. Les méthodes Acteur-Critiques sont toujours des méthodes par différence temporelle.

Algorithm 2.8 Acteur-Critique en-politique

$\theta \leftarrow$ initialisés aux choix de l'utilisateur
$\phi \leftarrow$ initialisés aux choix de l'utilisateur
procedure Acteur-Critique(η_θ, η_ϕ)
    loop
        Initialiser s
        $I \leftarrow 1$
        while s n'est pas terminal **do**
            $a \leftarrow \pi_\theta(s)$
            $s', r \leftarrow env(a)$
            $\mathcal{A}_t \leftarrow r + \gamma V_\phi(s') - V_\phi(s)$
            $\theta \leftarrow \theta + \eta_\theta \mathcal{A}_t \nabla \log \pi_\theta(a_t|s_t)$
            $\phi \leftarrow \phi - \eta_\phi \mathcal{A}_t \nabla V_\phi(s_t)$
            $I \leftarrow \gamma I$
            $s \leftarrow s'$

2.3.4 Acteur-Critique avec région de confiance

Pour continuer à améliorer la stabilité de l'entraînement, les méthodes acteur-critiques par *région de confiance* ont été proposées. Les algorithmes d'*optimisation de politiques par région de confiance* (Trust Region Policy Optimization, TRPO) [121] proposent de limiter l'amélioration de la politique à chaque étape pour éviter qu'elle ne diverge catastrophiquement.

L'échantillonnage préférentiel permet d'estimer l'espérance μ d'une variable aléatoire $f(x)$ sous une distribution p en ayant des échantillons provenant d'une autre distribution q tel que

$$\mathbb{E}_{s\sim q}\Big[\frac{p(x)}{q(x)} f(x)\Big] = \mu. \tag{2.30}$$

Les méthodes de gradient de politique *hors-politique* [43] se servent souvent de l'échantillonage préférentiel afin d'estimer l'espérance du retour selon la politique *cible* en utilisant des échantillons provenant de la politique *comportementale* β pour obtenir le gradient de la politique *cible* tel que

$$\nabla J(\theta) = \mathbb{E}_{s,a\sim\beta}\Big[\frac{\pi_\theta(a|s)}{\beta(a|s)} q(s,a) \nabla \log \pi_\theta(a|s)\Big]. \tag{2.31}$$

Algorithm 2.9 Proximal Policy Optimization

$\theta, \theta_{old} \leftarrow$ initialisés aux choix de l'utilisateur
$\phi, \phi_{old} \leftarrow$ initialisés aux choix de l'utilisateur
procedure PPO($\eta_\theta, \eta_\phi, \epsilon, K$)
    loop
        Générer des trajectoires $D = \{\tau_0, \tau_1, ..., \tau_n\}$ en suivant $\pi_{\theta_{old}}(\cdot|\cdot)$
        Calculer le retour R $\leftarrow \sum_t^{|\tau_i|} r_t^{t\gamma} \; \forall \; r_t \in \tau_i \; \forall \; \tau_i \in D$
        Calculer l'avantage $\mathcal{A}_{\phi_{\text{old}}} \leftarrow r_t + \gamma V_{\phi_{\text{old}}}(s_{t+1}) - V_{\phi_{\text{old}}}(s) \; \forall \; (s_t, r_t, s_{t+1}) \in D$
        for k..K **do**
            $\rho \leftarrow \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \; \forall \; s, a \in D$
            $\theta \leftarrow \theta + \eta_\theta \nabla \min\left(\rho \mathcal{A}_{\phi_{\text{old}}}, \text{clip}(\rho, 1-\epsilon, 1+\epsilon)\mathcal{A}_{\phi_{\text{old}}}\right)$
            $\phi \leftarrow \phi - \eta_\phi \nabla (V_\phi(s_t) - R) \; \forall \; s \in D$
        $\theta_{\text{old}} \leftarrow \theta$
        $\phi_{\text{old}} \leftarrow \phi$

TRPO propose, un peu comme DQN, d'utiliser une version précédente de la politique cible en tant que politique comportementale. Ceci est spécialement utile lors de l'utilisation de plusieurs acteurs échantillonnant l'environnement en parallèle et que la mise à jour de ceux-ci peut être coûteuse. Dans ce cas-ci, la politique comportementale est notée $\pi_{\theta_{\text{old}}}$ et l'objectif est formulé en tant que :

$$J(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_{\text{old}}}}\Big[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\mathcal{A}_{\phi_{\text{old}}}(s,a)\Big]$$
$$t.q.\; \mathbb{E}_{s\sim\pi_{\theta_{\text{old}}}}\Big[D_{KL}(\pi_{\theta_{\text{old}}}(\cdot|s)||\pi_\theta(\cdot|s))\Big] \leq \delta, \tag{2.32}$$

où la condition, calculant la divergence de Kullback-Leibler entre la "vieille" et la nouvelle politique, s'assurent que les deux restent similaires selon une constante δ. Cette contrainte force la nouvelle politique à rester dans une "région de confiance" autour de l'ancienne politique ; l'intuition étant qu'une nouvelle politique divergeant trop de l'ancienne ne permettra pas à l'ancienne de bien estimer la nouvelle. Par contre, calculer la condition de l'objectif de TRPO peut être compliqué et coûteux.

Afin de réduire la complexité des méthodes de gradient de politique avec région de confiance, l'algorithme d'*optimisation proximale de politique* (Proximal Policy Optimization, PPO) [129] tente de conserver la notion de "région de confiance" en

simplement tronquant le ratio entre les politiques tel que

$$J(\theta) = \min\Big(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\mathcal{A}_{\phi_{\text{old}}}(s,a), \text{clip}(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}, 1-\epsilon, 1+\epsilon)\mathcal{A}_{\phi_{\text{old}}}(s,a)\Big), \quad (2.33)$$

où la fonction $\text{clip}(x, a, b)$ force la valeur x à être entre les bornes a et b. L'objectif de PPO peut sembler complexe ; il est donc utile d'analyser les deux cas possibles :

$\mathcal{A}_{\phi_{\text{old}}}(\mathbf{s}, \mathbf{a}) > \mathbf{0}$

Si l'avantage est positif, le processus d'optimisation tentera d'augmenter $\pi_\theta(a|s)$. Par contre, la fonction $\min(a, b)$ forcera le ratio à demeurer en deçà de $1 + \epsilon$, gardant la nouvelle politique proche de l'ancienne.

$\mathcal{A}_{\phi_{\text{old}}}(\mathbf{s}, \mathbf{a}) < \mathbf{0}$

Si l'avantage est négatif, le processus d'optimisation tentera de réduire $\pi_\theta(a|s)$. Par contre, la fonction $\text{clip}(x, a, b)$ préviendra le ratio de descendre plus bas que $1 - \epsilon$, gardant toujours la politique proche de l'ancienne.

On peut constater en regardant l'algorithme 2.9 que PPO apporte quelques contributions de plus dans le but d'améliorer la stabilité de l'entraînement. Par exemple, l'entraînement se fait sur une *batch* d'échantillons plutôt que sur une seule transition. De plus, chaque *batch* est réutilisée K fois plutôt qu'une, minimisant le problème de la *cible mouvante* et réduisant par le fait-même la demande en échantillons. On peut aussi remarquer que PPO utilise les retours potentiellement complets (en se rappelant que $|\tau| \leq |\text{episode}|$), le plaçant à mi-chemin entre les méthodes par différence temporelle et les méthodes Monte Carlo.

2.3.5 Gradient de politique déterministe

Le lecteur attentif aura peut-être remarqué que les politiques apprises par les méthodes de gradient de politique et acteur-critiques sont des politiques stochastiques, où $\pi(a|s)$ dénote la *probabilité* d'effectuer l'action a dans l'état s. Même dans le cas d'actions discrètes, où la politique utilise la fonction *softmax*, les actions sont échantillonnées selon la distribution de probabilité donnée par le *softmax.* Pour apprendre la politique, une ascension de gradient sur une variante de l'équation

suivante est effectuée :

$$\nabla J(\theta) = \mathbb{E}_{s,a\sim\pi_\theta}\Big[\nabla \log \pi_\theta(a|s) q_\phi(s,a)\Big]. \tag{2.34}$$

Par contre, l'algorithme DQN (2.5) utilise la politique vorace *déterministe* suivante :

$$\pi_{\text{det}}(s) = \arg\max_a Q_\phi(s,a), \tag{2.35}$$

et tente de minimiser la fonction de perte suivante

$$\begin{aligned} L(\phi) &= \frac{1}{2}\Big(q_\phi(s_t,a_t) - u_t\Big)^2 \\ t.q.\ u_t &= r_t + \gamma q_{\phi'}(s_{t+1}, \pi_{\text{det}}(s_{t+1})). \end{aligned} \tag{2.36}$$

La politique déterministe $\pi_{\text{det}}(s)$ peut être définie par :

$$\pi_{\text{det}}(a|s) \begin{cases} 1 & \arg\max_a q_\phi(s,a) \\ 0 & \text{pour tout le reste.} \end{cases} \tag{2.37}$$

Il serait très difficile d'adapter l'algorithme DQN pour qu'il fonctionne sur un espace d'état continu puisque la politique requièrerait d'évaluer une infinité d'actions. Pour obtenir une politique déterministe, les algorithmes de gradient de politique déterministes [122] utilisent plutôt une politique paramétrée $\pi_{\text{det}_\theta}(s)$ (notée $\pi_\theta(s)$ pour le reste de ce manuscrit), comme dans le cas des méthodes à gradient de politique stochastique. L'objectif demeure de maximiser le retour espéré selon la politique, défini en tant que :

$$J(\theta) = \mathbb{E}_{s\sim\pi}[q(s,\pi(s))]. \tag{2.38}$$

Puisque nous avons une politique et une fonction action-valeur différentiable, le théorème de politique de gradient déterministe [122] dicte que performer une ascension de gradient sur la sortie de la fonction *action-valeur* permet de maximiser implicitement celle-ci à travers la politique. La mise à jour de la politique devient :

$$\nabla J(\theta) = \mathbb{E}_{s\sim\beta}[\nabla_\pi Q_\phi(s,\pi(s)) \nabla_\theta \pi_\theta(s)], \tag{2.39}$$

où la règle de dérivée en chaîne est appliquée afin de propager le gradient de la politique à travers la fonction action-valeur.

L'algorithme de *Gradient de politique profond déterministe* [88] (Deep Deterministic Policy Gradients, DDPG) propose d'utiliser des réseaux de neurones profonds pour approximer π et Q. De plus, DDPG reprend quelques idées de DQN en utilisant un tampon de mémoire échantillonné uniformément à chaque pas de temps et propose d'utiliser un réseau *cible* pour stabiliser l'autoamorçage. Par contre, DDPG propose d'effectuer une mise à jour progressive des poids du réseau cible en effectuant une forme de moyennage définie en tant que :

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi' \tag{2.40}$$

où $\tau \ll 1$, causant moins de changements brusques dans la cible. De plus, avoir une politique déterministe permet de découpler le problème de l'apprentissage de celui de l'exploration. De ce fait, une politique exploratoire π' est construite en ajoutant du bruit ϵ à la sortie de l'acteur, définie par :

$$\pi'(s) = \pi_\theta(s) + \epsilon,\ \epsilon \sim \mathcal{N}(\mu, \sigma^2). \tag{2.41}$$

Toujours dans le but de stabiliser l'apprentissage, une politique cible $\pi_{\theta'}$ est aussi utilisée lors du calcul de u_t, et ses poids sont mis à jour de façon similaire à ceux de la fonction *action-valeur*.

Même si DDPG a produit des résultats étonnants sur la suite OpenAI Gym [8], quelques problèmes persistent. Notamment, le critique de DDPG tend à surestimer les valeurs d'états, menant à des états indésirables ayant une valeur désirable et forçant la politique à avoir un comportement sous-optimal. De plus, puisque la cible de la fonction *action-valeur* dépend de la politique, la mise à jour de la politique peut rendre la fonction Q instable. Finalement, la politique peut parfois exploiter des erreurs d'estimation de la fonction Q menant encore à des performances sous-optimales.

L'algorithme *Gradient de politique déterministe jumelé et différé* (Twin-Delayed Deep Deterministic Policy Gradient, TD3) [47] aborde ces problèmes de trois façons : Pour éviter la surestimation des valeurs d'états, TD3 emprunte le concept

Algorithm 2.10 Twin Delayed Deep Deterministic Policy Gradient

$\theta, \theta_{\text{cible}} \leftarrow$ initialisés aux choix de l'utilisateur
$\phi_1, \phi_2, \phi_{1_{\text{cible}}}, \phi_{2_{\text{cible}}} \leftarrow$ initialisés aux choix de l'utilisateur
$D = \{\}$
procedure TD3($\eta_\theta, \eta_\phi, \sigma, \sigma_{\text{cible}}, \tau$)
    loop
        Initialiser s
        while s n'est pas terminal **do**
            $a \leftarrow \pi_\theta(s) + \mathcal{N}(0, \sigma)$
            $r, s' \leftarrow env(a)$
            $D \leftarrow D \cup (s, a, r, s')$
            échantillonner de D une mini-batch de transitions (S, A, R, S')
            $A' \leftarrow \pi_{\theta_{\text{cible}}}(S) + \mathcal{N}(0, \sigma_{\text{cible}})$
            $U \leftarrow R + \gamma min_{i=1,2} Q_{\phi i_{\text{cible}}}(S', A')$
            $\phi_i \leftarrow \phi_i - \eta_\phi \Big[U - Q_{\phi_i}(S, A)\Big] \nabla Q_{\phi_i}(S, A) \; \forall \; i \in 1, 2$
            if il est temps de mettre à jour la politique **then**
                $\theta \leftarrow \theta + \eta_\theta \nabla_\pi Q_\phi(s, \pi(s)) \nabla_\theta \pi_\theta(s)$
                $\theta_{\text{cible}} \leftarrow \tau\theta + (1 - \tau)\theta_{\text{cible}}$
                $\phi_{\text{cible}_i} \leftarrow \tau\phi_i + (1 - \tau)\phi_{\text{cible}_i} \; \forall \; i \in 1, 2$
            $s, a \leftarrow s', a'$

d'*Apprentissage Q double* [58] en ayant deux fonctions Q apprises simultanément, et en prenant le minimum des deux valeurs lors de la mise à jour de celles-ci. TD3 met aussi à jour la politique moins fréquemment que les fonctions Q, dans le but de stabiliser l'apprentissage. Finalement, TD3 ajoute du bruit à l'*action cible* dans le but d'adoucir les changements dans la valeur d'un état selon les actions.

2.3.6 Maximisation de l'entropie via un acteur stochastique

L'utilisation d'une politique déterministe par TD3 apporte son lot de problèmes : Notamment, l'algorithme doit explicitement ajouter du "bruit" aux actions choisies lors de la phase de récolte de transitions afin d'explorer l'environnement. De plus, les politiques déterministes convergent forcément vers une solution optimale singulière alors qu'il existe possiblement une famille de politiques optimales, par exemple lorsqu'il y a de l'incertitude dans l'environnement. Il serait donc préférable de promouvoir l'apprentissage de politiques ayant une grande stochasticité. À cet effet, le cadre

Algorithm 2.11 Soft Actor-Critic

$\theta \leftarrow$ initialisé aux choix de l'utilisateur
$\phi_1, \phi_2, \phi_{1_{\text{cible}}}, \phi_{2_{\text{cible}}} \leftarrow$ initialisés aux choix de l'utilisateur
$D = \{\}$
procedure SAC($\eta_\theta, \eta_\phi, \alpha, \tau$)
    loop
        Initialiser s
        while s n'est pas terminal **do**
            $a \leftarrow \pi_\theta(s)$
            $r, s' \leftarrow env(a)$
            $D \leftarrow D \cup (s, a, r, s')$
            échantillonner de D une mini-batch de transitions (S, A, R, S')
            $A' \leftarrow \pi_\theta(S')$
            $\hat{A} \leftarrow \pi_\theta(S)$
            $U \leftarrow R + \gamma min_{i=1,2} Q_{\phi i_{\text{cible}}}(S', A') - \alpha \log(\pi(A'|S'))$
            $\phi_i \leftarrow \phi_i - \eta_\phi \Big[U - Q_{\phi_i}(S, A)\Big] \nabla Q_{\phi_i}(S, A) \; \forall \; i \in 1, 2$
            $\theta \leftarrow \theta + \eta_\theta \nabla_\theta \alpha \log(\pi_\theta(\hat{A}|S)) + (\nabla_{\hat{A}} \alpha \log(\pi_\theta(\hat{A}|S)) - \nabla_{\hat{A}} Q_\phi(S, \hat{A}))$
            $\phi_{\text{cible}_i} \leftarrow \tau \phi_i + (1 - \tau) \phi_{\text{cible}_i} \; \forall \; i \in 1, 2$
            $s, a \leftarrow s', a'$

d'*apprentissage par renforcement à entropie maximale* [65] a été introduit afin de permettre l'apprentissage de politiques maximisant à la fois le retour escompté et leur propre entropie.

L'entropie d'une variable aléatoire (discrète) correspond à la "surprise" ou "l'incertitude" relative aux valeurs possibles que celle-ci peut prendre, et est définie par :

$$\mathbb{H}(X) = -\sum_{i=0}^{N} p(x_i) \log(p(x_i)), \tag{2.42}$$

avec x_i les valeurs possibles que peut prendre la variable aléatoire X. Si on formule la politique apprise par les agents comme étant une variable aléatoire $\pi(s)$ dont les valeurs $a_0, ...a_n$ sont les actions possibles à l'état s, on peut facilement voir qu'une politique déterministe aura une entropie minimale, puisque la politique attribuera une probabilité de 100% à l'action apprise et 0% aux autres.

Pour palier à ce problème, le cadre d'*apprentissage par renforcement à entropie maximale* modifie alors l'objectif typique de l'apprentissage par renforcement afin d'y

introduire l'entropie de la politique apprise. Formellement, on souhaite trouver la politique optimale définie par :

$$\pi^* = \arg\max_{\pi} \sum_t \mathbb{E}_{(s_t,a_t)\sim\pi}\Big[r_t + \alpha\mathbb{H}(\pi(s_t))\Big], \tag{2.43}$$

où α est un paramètre de température permettant de retrouver l'objectif "standard" de l'apprentissage par renforcement. Bâtissant sur ce principe (et sur DDPG), l'algorithme *Soft Acteur-critique* (Soft Actor-Critic, SAC) [66] a été introduit afin d'entraîner des politiques stochastiques à entropie maximale. Bien que SAC partage quelques similarités avec TD3 (l'utilisation du concept d'apprentissage Q double, de moyennage des paramètres des réseaux cibles et d'*experience replay*), celui-ci a été introduit parallèlement à TD3.

Afin d'entraîner les réseaux Q_ϕ avec l'objectif reformulé, SAC emploie une fonction de perte prenant la forme

$$\begin{aligned} L(\phi) &= \mathbb{E}_{(s,a,r,s')\sim D}\Big[U - \Big(Q_\phi(s,a)\Big)^2\Big] \\ t.q.\ U &= r + \gamma(Q_{\phi'}(s',a') - \alpha\log(\pi_\theta(a'|s'))),\ a'\sim\pi(s'), \end{aligned} \tag{2.44}$$

suivant la définition de l'entropie pour une variable aléatoire continue définie par :

$$\mathbb{H}(X) = \mathbb{E}[I[X]] = \mathbb{E}_{x\sim X}[-\log(f(x))], \tag{2.45}$$

$f(x)$ étant la fonction de masse de X. Similairement, l'objectif de la politique devient : [5]

$$J(\theta) = \mathbb{E}_{s\sim\pi}[\log(\pi_\theta(a|s)) - Q_\phi(s,a)],\ a\sim\pi(s'), \tag{2.46}$$

5. La raison exacte de la forme de cette fonction de perte ainsi que les étapes de dérivations y menant sont disponibles dans l'article introduisant l'algorithme SAC [65].

menant à la mise à jour suivante :

$$\nabla J(\theta) = \mathbb{E}_{s\sim D}\Big[\nabla_\theta \log(\pi_\theta(a|s)) \; + \; \Big(\nabla_\pi \log(\pi_\theta(a|s)) \\ - \; \nabla_\pi Q_\phi(s, \pi_\theta(s))\Big)\nabla_\theta \pi_\theta(s)\Big], a \sim \pi\theta(s), \quad (2.47)$$

où le paramètre α a été omis à des fins de clarté. L'algorithme complet est disponible dans l'encadré 2.11.

2.4 Conclusion

Dans ce chapite, nous avons fait un survol de la théorie derrière l'apprentissage par renforcement et son extension : L'apprentissage par renforcement profond. Nous avons défini le contexte du processus de décision markovien, nous avons exploré des algorithmes "classiques" ayant de bonnes propriétés de convergences, mais étant inutilisables dans le monde réel. Nous avons ensuite exploré l'extension de l'apprentissage par renforcement utilisant des réseaux de neurones afin d'apprendre la valeur d'un état ou simplement l'action à effectuer à celle-ci. Dans les prochains chapitres, nous verrons comment utiliser l'apprentissage par renforcement profond pour attaquer le problème de la tractographie, où les vérités terrains ne sont pas disponibles.

Singularity is in the eye of the beholder

—Peter Norvig

Chapitre 3

Tractographie

L'entièreté de l'expérience humaine est due à son système nerveux : sans lui, nous ne serions pas bien plus réactifs qu'un concombre de mer. Un organe extrêmement complexe et la partie centrale du système nerveux humain, le cerveau est responsable entre autre de gérer, analyser et répondre à tous les stimuli que l'être humain éprouve au long de sa vie.

Comprendre le fonctionnement exact du cerveau humain demeure un objectif à long terme. Récemment, le connectome complet du *Caenorhabditis elegans*, un petit ver d'environs un millimètre de long, a été péniblement reconstruit afin d'identifier la fonction de chacune de ses 302 neurones et les 7000 connections entre-elles [26]. Cette avancée est incroyable, car il s'agit du premier connectome complet d'un animal. Par contre, l'être humain possède environs 86 millions de ces neurones et des ordres de magnitudes plus de synapses encore, et l'obtention d'un connectome complet humain demeure chose lointaine.

La tractographie permet d'étudier la structure de la matière blanche de l'être humain *in-vivo* en extrayant des informations directionnelles du signal provenant de l'imagerie par résonance magnétique de diffusion. Grâce à la tractographie, il nous est possible d'avoir une représentation virtuelle des connections physiques présentes dans le cerveau humain. La connectomique, soit l'étude des connections du système nerveux, dépend grandement de la tractographie afin de faire le pont entre les informations

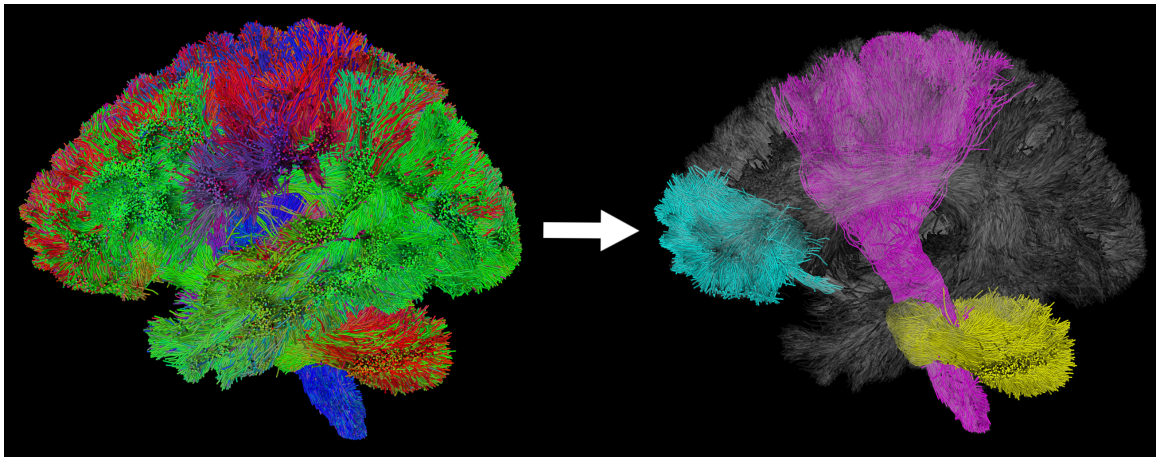

Figure 3.1 – Illustration d'un tractogramme, soit la sortie d'un algorithme de tractographie, puis la dissection virtuelle en faisceaux.

fonctionnelles et structurelles du cerveau humain [123].

Dans ce chapitre, nous décrirons ce qu'est la tractographie. Pour s'effectuer, celle-ci a besoin d'une représentation de l'orientation des fibres de la matière blanche. Donc, avant même de décrire la tractographie, nous explorerons d'abord les façons d'obtenir ces orientations. Nous ferons d'abord un survol de différents types d'imagerie basés sur l'imagerie par résonance magnétique : nous commencerons par faire une brève description de l'anatomie du cerveau humain, puis nous ferons une revue superficielle du fonctionnement de l'imagerie par résonance magnétique, l'imagerie par résonance magnétique de diffusion puis son utilisation pour l'imagerie par tenseur de diffusion et l'imagerie de diffusion avec haute résolution angulaire. Ensuite, nous aborderons le contexte de la tractographie : comment reconstruire de façon virtuelle les fibres de la matière blanche, différents algorithmes pour y procéder, les problèmes connus avec ceux-ci, quelques améliorations possibles au processus de tractographie, comment valider que les algorithmes font bien leur travail et finalement, un survol de la tractographie par apprentissage machine.

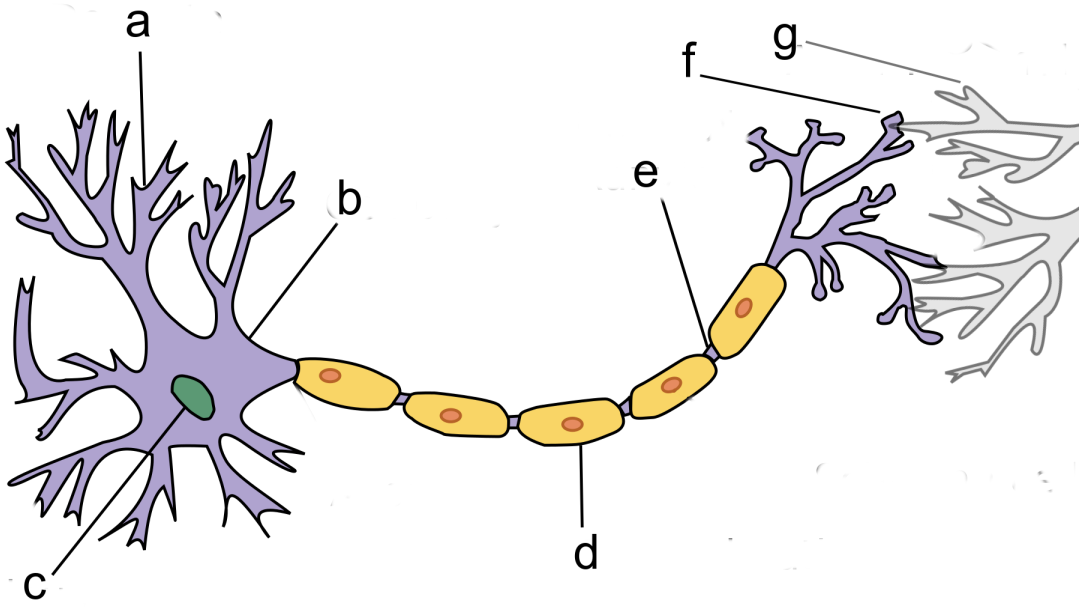

Figure 3.2 – Schéma des composantes d'un neurone.
a) Dendrites. b) Soma. c) Noyau. d) Gaine de myéline. e) Axone. f) Terminaisons axonales, à leurs extrémités les synapses. g) Dendrites d'un autre neurone. Adapté de Wikimedia Commons.

3.1 Neuroimagerie par résonance magnétique

Le cerveau humain est un organe fascinant. Ne représentant que 2% de la masse d'un adulte, il consomme 20% de l'énergie produite par le corps. Afin d'en sonder la structure, l'imagerie par résonance magnétique a été développée et celle-ci permet d'obtenir des images par contrastes de façon non-invasive. Dans cette section, nous ferons un survol de l'anatomie du cerveau ainsi qu'un aperçu des différentes techniques d'imagerie par résonance magnétique.

3.1.1 Structure du cerveau

Deux types de cellules composent le cerveau humain : les cellules gliales et les neurones. Les cellules gliales assurent un rôle de support aux neurones, tant structurel que nutritionnel. De plus, elles sont responsables de la production de myéline, la circulation du sang dans le cerveau et la transmission synaptique [144].

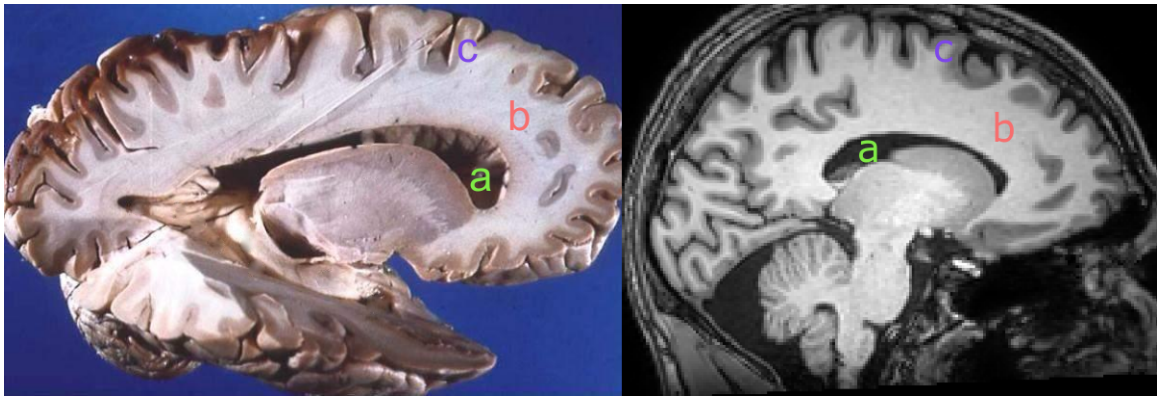

Figure 3.3 – La dissection en coupe sagitale, à gauche, permet de bien apprécier le contraste entre la matière grise, ici rose-foncé, se trouvant surtout à la surface du cortex, et la matière blanche. À droite, un tranche sagitale plus-ou-moins équivalente d'une image par contraste *T1*. a) 3e ventricule b) Matière blanche c) Matière grise.

Les neurones (c.f. fig. 3.2), quant à eux, sont les cellules responsables de traiter et d'acheminer l'information dans le cerveau en se connectant les unes aux autres et en formant un réseau des plus impressionnants. Les neurones sont composés de quatre parties distinctes : le soma, les dendrites, l'axone et les synapses. Le soma, ou corps cellulaire, est la partie centrale du neurone, d'où émerge les dendrites et l'axone. Les dendrites, les points d'entrée de la neurone, reçoivent les signaux provenant des synapses d'autres neurones et l'envoient vers le soma. L'axone, long câble d'environ un micromètre de diamètre, émergeant du soma, sert à transférer l'information du corps cellulaire vers les autre neurones à travers les synapses, au bout des terminaisons axonales.

Grossièrement, le cerveau humain peut être divisé en trois composantes : la matière grise (MG), la matière blanche (MB) et le liquide cérébrospinal (LCS) (c.f. fig. 3.3). La matière grise est le centre de traitement de l'information et est composée principalement de soma, dendrites, et cellules gliales ainsi que de quelques axones. La matière grise se retrouve principalement à la surface du cerveau et du cervelet, ainsi que dans quelques régions à l'intérieur du cerveau telles le thalamus et quelques nucléus. La matière blanche, quant à elle, est composée très majoritairement d'axones et sert à relier les différentes parties de la matière grise. Ces axones, ayant une longueur pouvant varier entre un millimètre et un mètre, tendent à être regroupés ensemble et à former un

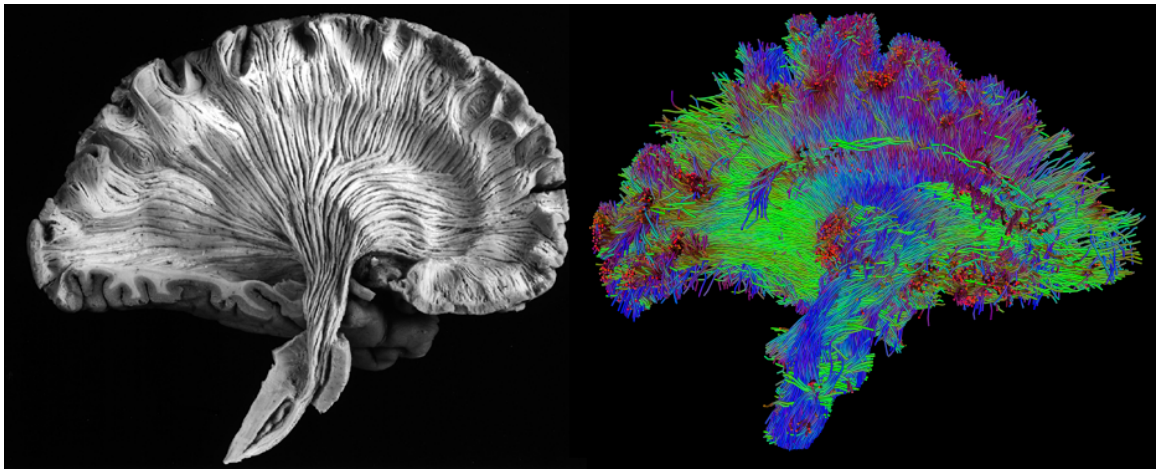

Figure 3.4 – L'image de gauche permet de bien apprécier la structure fibreuse et dirigée de la matière blanche tandis que l'image de droite présente les tracts virtuelles associées. Gauche) Dissection de la matière blanche, où le *corona radiata*, la radiation thalamique antérieure et les fibres cortico-spinales et pyramidales de l'hémisphère gauche sont mises en évidence. Droite) Dissection virtuelle d'un tractogramme des tracts équivalentes. Adapté de [153].

ensemble structuré que nous appellerons «tracts», ou *bundle* (faisceau), dont certaines sont visibles à la figure 3.4.

L'étude de la structure de la matière blanche a plusieurs applications possibles : la sclérose en plaques, par exemple, cause des lésions menant à une démyélinisation des axones en ces endroits [61]. Une diminution du volume de la matière blanche peut être associée à des pertes de fonctions cognitives et motrices, telle la maladie d'Alzheimer [103]. De plus, une connaissance de la disposition des différents faisceaux de la matière blanche est un atout immense pour la planification et le déroulement de neurochirurgies [48].

3.1.2 Imagerie par résonance magnétique

L'imagerie par résonance magnétique (IRM) est une technique d'imagerie permettant d'obtenir des images anatomiques de façon non-invasive grâce à de puissants champs magnétiques. Les tissus du corps humain contiennent tous une quantité variable d'atomes d'hydrogènes, ayant pour nucléus qu'un seul proton. En temps normal, les

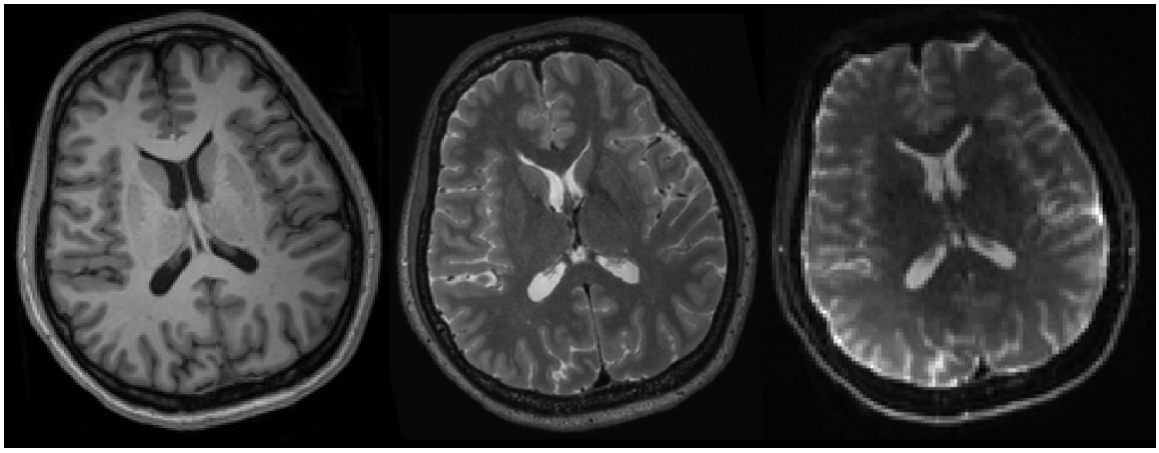

Figure 3.5 – Comparaison de différentes séquences d'acquisitions, à la même tranche axiale du même sujet, provenant du jeux de données *Human Connectome Project* MGH-USC [49]. a) Image tirée du séquence T1w, acquise à une résolution de 1mm isotropique. b) Image tirée d'une séquence T2w, acquise à une résolution de 0.7mm isotropique. c) Image b0 tirée d'une séquence de diffusion acquise à une résolution de 1.5mm isotropique.

protons se déplacent dans des directions aléatoires créant un signal électromagnétique nul. Lorsque des tissus sont placés dans un scanneur, les protons de ceux-ci deviennent magnétisés selon une direction fixe appelée *spin*. L'alignement des spins crée un signal électromagnétique mesurable. Appliquer un champs de radio-fréquences permet de changer la direction de la magnétisation des tissus présent, et le relâchement du champs de radio-fréquences permet aux tissus de retourner à leur orientation d'origine.

Par contre, différents tissus retournent plus-ou-moins rapidement à leur direction d'origine (généralement influencé par la concentration de molécules d'eau dans le tissu, celles-ci riches en protons), et le changement de magnétisme variable entre les tissus peut être mesuré pour obtenir des contrastes dans l'image reconstruite. Dans le but de pouvoir différencier les différentes parties de l'image, celle-ci est acquise en "tranches" et un gradient (plutôt qu'un champs uniforme) de champs magnétique est utilisé pour cette tranche attribuant une fréquence et une phase différente pour chaque pixel de la tranche reconstruite [125]. La reconstruction subséquente des tranches de l'image en volume 3D donne lieu au terme *voxel* (*volumetric pixel*) pour décrire les éléments du volume reconstruit.

Diverses propriétés du changement de la magnétisation des tissus peuvent être observées à travers différentes séquences d'acquisition. Les séquences *T1* et *T2* (c.f. fig. 3.5), par exemple, obtiennent des contrastes inversés pour la matière blanche, la matière grise et le LCS [102].

3.1.3 Imagerie par résonance magnétique de diffusion

L'imagerie par résonance magnétique de diffusion (IRMd) est basée sur le mouvement Brownien des molécules d'eau à l'intérieur et l'extérieur des cellules du cerveau humain [81, 84]. Le mouvement Brownien, étudié par Einstein en 1905 [44], estime la probabilité d'un déplacement x d'une molécule d'eau au temps t t.q :

$$p(x,t) = \frac{1}{\sqrt{4\pi t D}} e^{-\frac{x^2}{4tD}}, \tag{3.1}$$

qui est en fait une gaussienne moyennée à 0 et ayant une variance de $2Dt$, où D est un scalaire représentant le coefficient de diffusion : la latitude qu'a la molécule d'eau à se déplacer librement, sans restriction, selon son environnement.

L'IRMd tente d'estimer ce coefficient de diffusion en présupposant qu'il n'est pas uniforme partout dans le cerveau ; après tout, il serait logique de penser qu'une molécule d'eau a plus de facilité à se déplacer dans les ventricules, où se retrouve beaucoup le LCS, que dans la matière blanche, une partie hautement structurée du cerveau. Pour ce faire, un gradient de champs magnétiques est appliqué dans le but de changer la phase des molécules d'eau. Les molécules sont laissées à elles-même pendant un certain *temps de diffusion*, puis le gradient inverse est appliqué afin de retourner les molécules d'eau à leur phase originelle.

Une molécule d'eau très restreinte dans ses mouvements (ayant un coefficient de diffusion bas) ne bougera que très peu lors du temps de diffusion. Donc, lorsque le gradient inverse sera appliqué à celle-ci, elle reviendra à sa phase d'origine et très peu ou aucune perte d'intensité de signal ne sera observé. Par contre, une molécule d'eau ayant un coefficient de diffusion plus grand aura pu se déplacer lors du temps de diffusion. Lors de l'application du gradient inverse, celle-ci recevra une transformation différente que celle nécessaire à son rephasage. Puisque la molécule d'eau se retrouve alors déphasée par rapport aux autres molécules avoisinantes, une perte de signal peut

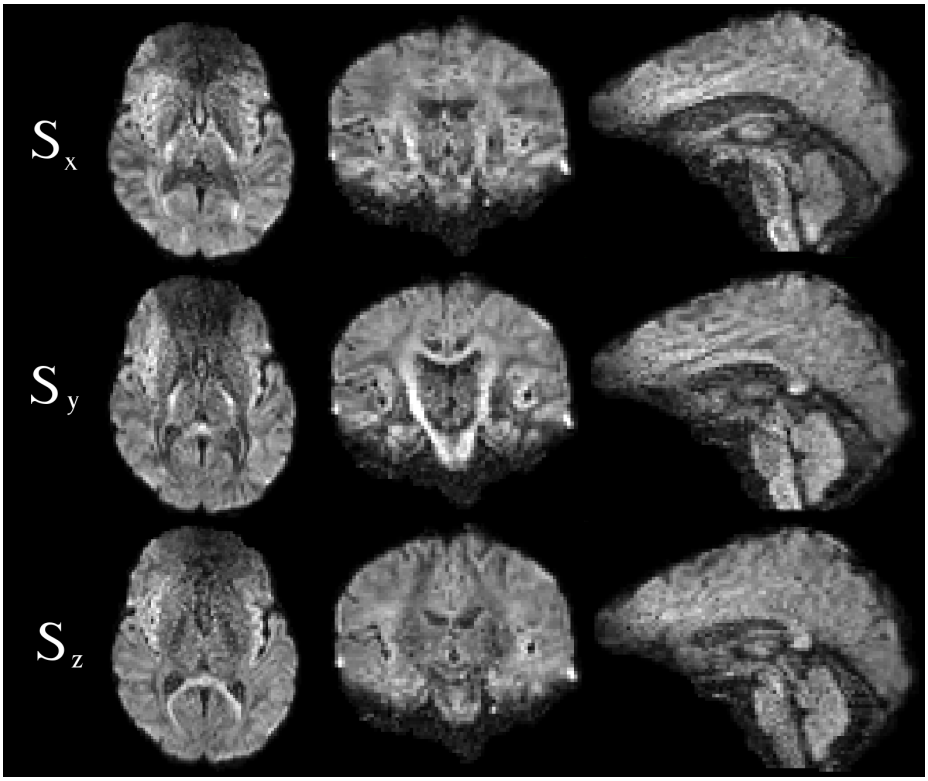

Figure 3.6 – Le signal de diffusion selon un gradient en x, y ou z, acquis à une résolution de 2mm isotropique et $b = 1000$. S_x représente un gradient de diffusion sur l'axe des x. On peut s'apercevoir que les même structures ont des intensités différentes selon la direction du gradient avec lequel l'image a été acquise. Cette variation, surtout présente dans la matière blanche, viens du fait que la nature structurée de celle-ci permet plus facilement le déplacement des molécules d'eau dans certaines directions.

être observé.

L'équation du signal de diffusion en un voxel x, y, z peut être défini en tant que

$$S(b) = S_0 e^{-bD}, \tag{3.2}$$

où D est le diffusion au voxel, b un terme regroupant la force du gradient et le temps de diffusion et S_0 le signal sans diffusion ($b = 0$). Inversement, on peut retrouver le coefficient de diffusion apparent D (*apparent diffusion coefficient*, ADC) en chaque

voxel x, y, z grâce à

$$D_g = \frac{-1}{b}\log\Big(\frac{S}{S_0}\Big), \tag{3.3}$$

avec g le vecteur 3D représentant la direction du gradient appliqué. L'image résultante est nommée *image pondérée par diffusion* (*diffusion-weighted image*, DWI) [83] .

Par contre, ne mesurer la pondération de diffusion que pour une seule direction de gradient n'informe que sur la diffusion en cette direction. Il serait logique de penser que la diffusion des molécules d'eau dans le cerveau soit dépendante de l'orientation des structures avoisinantes. L'équation 3.1 présuppose une diffusion isotropique des molécules d'eau dans le cerveau, c'est à dire un mouvement uniforme dans toutes les directions, en modélisant la diffusion par un scalaire. Or, la nature hautement structurée de la matière blanche laisse présager que les molécules d'eau pourraient avoir plus de facilité à se déplacer le long des axones que perpendiculairement à ceux-ci. La figure 3.6 permet de visualiser l'intensité du signal pour certaines structures selon la direction du gradient appliqué. La *pseudo-ADC*, prenant en compte le signal selon le gradient dans les trois directions orthogonales x, y, z, peut être calculé comme étant

$$D = \frac{D_x + D_y + D_z}{3} \tag{3.4}$$

3.1.4 Imagerie par tenseur de diffusion

En 1994, Basser et. al [14] proposent de modéliser la diffusion de façon *anisotropique* en exprimant D plutôt comme étant un tenseur de taille 3×3 :

$$\mathbf{D} = \begin{pmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{yx} & D_{yy} & D_{yz} \\ D_{zx} & D_{zy} & D_{zz} \end{pmatrix}.$$

En intégrant le tenseur de diffusion $\mathbf{D}$ à l'équation 3.2, nous obtenons

$$S(b)_g = S_0 e^{-bg^T \mathbf{D} g}, \tag{3.5}$$

où g est un vecteur unitaire indiquant la direction du gradient. Par contre, puisque la

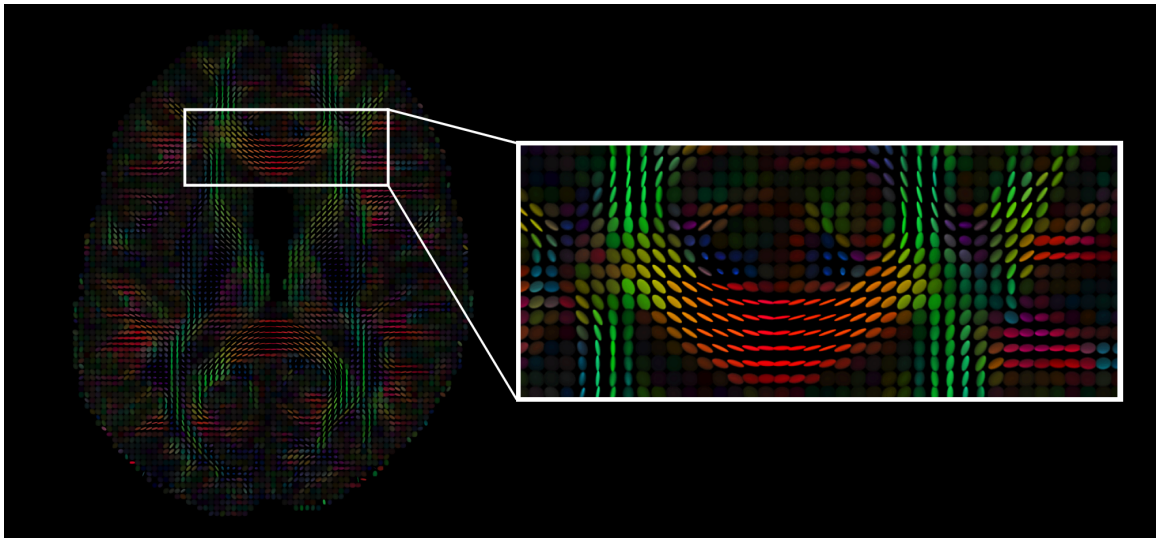

Figure 3.7 – Champs d'ellipsoïdes suite à la reconstruction des tenseurs à partir de l'information de diffusion. La couleur représente l'orientation de l'ellipsoïde (rouge pour gauche-droite, vert pour antérieur-postérieur et bleu pour inférieur-supérieur).

diffusion est supposée symétrique (donc qu'une molécule d'eau a autant de facilité à se déplacer dans une direction que son opposée, par exemple vers le haut d'un axone que vers le bas), nous obtenons l'équivalence $D_{xy} = D_{yx}$. L'équation 3.5 n'a alors que six inconnues plutôt que neuf, et donc n'a besoin que d'un minimum de six images acquises selon une direction de gradient g différente pour être définie. Ces six directions sont acquises en appliquant des gradients dirigés correspondants lors de l'acquisition de l'image de diffusion. Le vecteur $\mathbf{D} = [D_{xx}, D_{yy}, D_{zz}, D_{xy}, D_{xz}, D_{yz}]$ peut ensuite être calculé via une méthode des moindre-carrés [13].

Il est fréquent de représenter le tenseur $\mathbf{D}$ par ses vecteurs propres e_1, e_2, e_3 et valeurs propres $\lambda_1, \lambda_2, \lambda_3$, ainsi que de visualiser cette décomposition sous la forme d'un ellipsoïde tel qu'illustré dans la figure 3.7, donnant lieu au concept d'imagerie par tenseur de diffusion (*diffusion-tensor imaging*, DTI)

Plusieurs métriques peuvent être extraites de la décomposition de la matrice $\mathbf{D}$ en vecteurs et valeurs propres. Par exemple, l'anisotropie fractionnelle (*fractional anisotropy*, FA) [16], permettant de quantifier le niveau d'anisotropie à un voxel, est

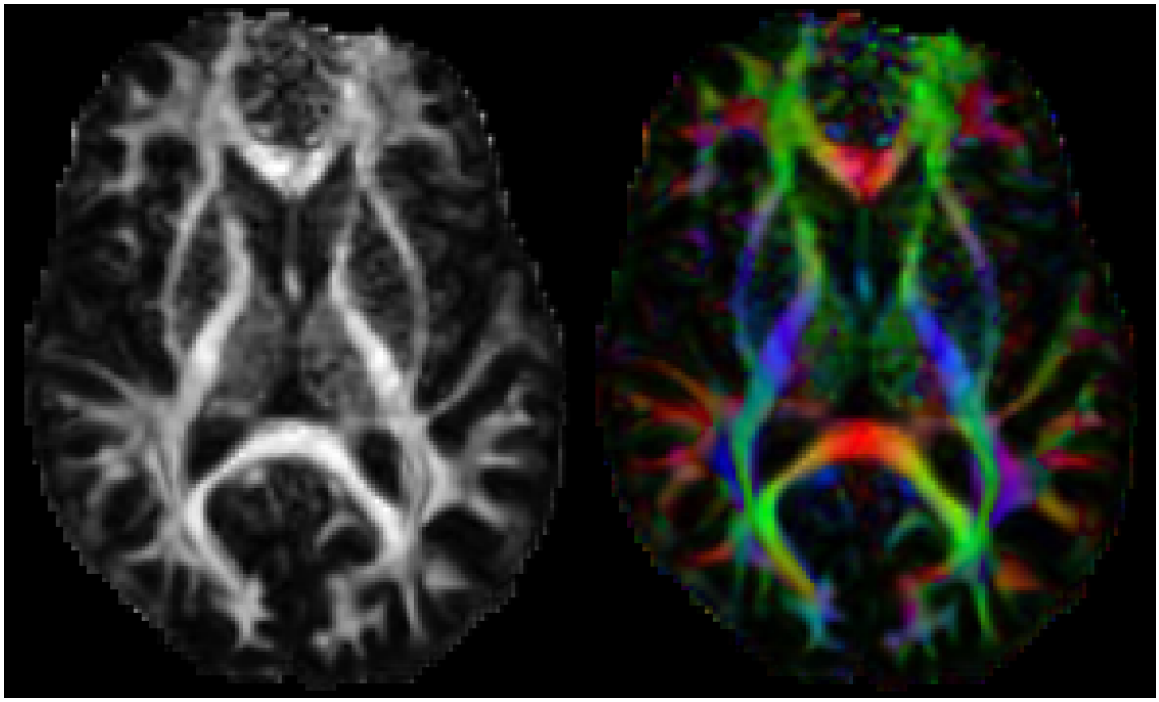

Figure 3.8 – Carte d'anisotropie fractionnelle extraite de l'imagerie par tenseurs de diffusion. À gauche : la carte de FA. À droite : la carte de FA, colorée selon l'orientation des tenseurs sous-jacents.

définie par

$$FA = \sqrt{\frac{3}{2}}\sqrt{\frac{(\lambda_1 - \bar{\lambda})^2 + (\lambda_2 - \bar{\lambda})^2 + (\lambda_3 - \bar{\lambda})^2}{\lambda_1^2 + \lambda_2^2 + \lambda_3^2}}, \tag{3.6}$$

avec $\bar{\lambda}$ la moyenne des valeurs propres. La figure 3.8 permet de visualiser une carte de FA extraite de l'imagerie par tenseurs de diffusion, ainsi qu'une carte de FA colorée par la direction des tenseurs sous-jacents.

Par contre, l'imagerie par tenseur de diffusion a ses problèmes. Par exemple, la modélisation de la diffusion par une gaussienne présuppose une seule population de fibres, alors qu'il est estimé qu'entre deux-tiers [36] et 90% [72] des voxels de la matière blanche contiennent plus d'une population de fibres. Dans le cas d'un croisement de fibres, l'ellipsoïde du tenseur de diffusion devient plutôt un "patatoïde" indifférenciable à une région à diffusion élevée, tel qu'illustré par la figure 3.9.

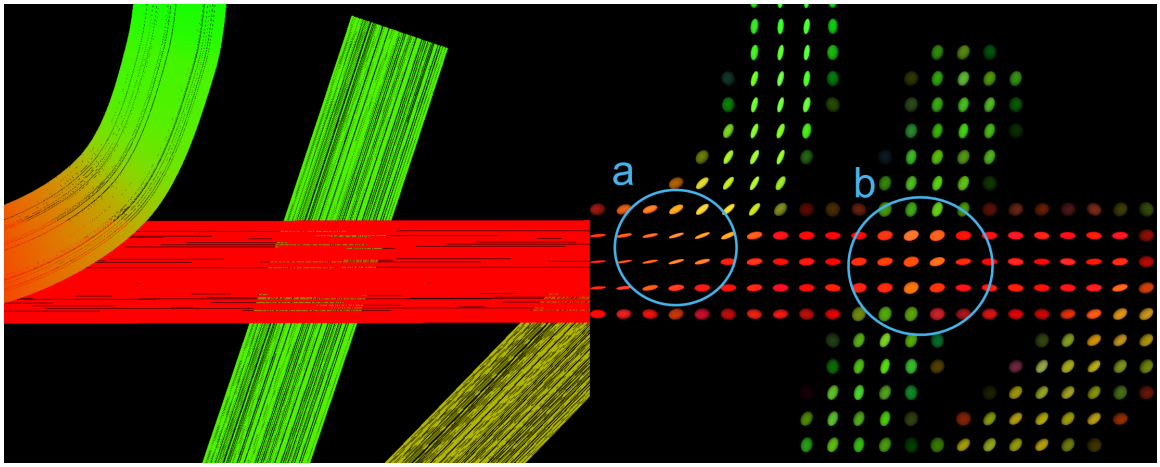

Figure 3.9 – Illustration d'un croisement de fibre et des tenseurs estimés. Nous pouvons observer que les tenseur estimés échouent à capturer les croisements. Pire encore, les tenseurs estimés dans la région a) indiquent une forte anisotropie, ce qui pourrait laisser présager une population singulière de fibres si la vérité terrain, à gauche, n'était pas disponible.

3.1.5 Imagerie de diffusion à haute résolution angulaire

Afin de palier aux problèmes de l'imagerie par tenseurs de diffusion, plusieurs méthodes tombant sous l'ombrelle de l'imagerie de diffusion à haute résolution angulaire (*High Angular Resolution Diffusion Imaging*, HARDI) ont été développées. Les méthodes HARDI peuvent être séparées en deux grandes familles : *avec-modèle* et *sans-modèle*. Les méthodes avec-modèle, comme la méthodes *Balle-et-bâton* (*Ball-and-stick* [159]), tentent de représenter le signal de diffusion en présumant une structure arbitraire aux voxels de la matière blanche. Les méthodes sans-modèle, au contraire, ne font aucune présomption et tentent de représenter la diffusion telle quelle.

Dans cette sections, nous nous concentrerons sur l'Imagerie par Q-Ball (*Q-Ball Imaging*, QBI) [143] ainsi que l'amélioration de celle-ci grâce à la déconvolution sphérique [135].

Permettons-nous d'introduire quelques notions (applicables à toutes techniques de représentation du signal de diffusion dites *sans-modèles* (*model-free*) [37]) : il est

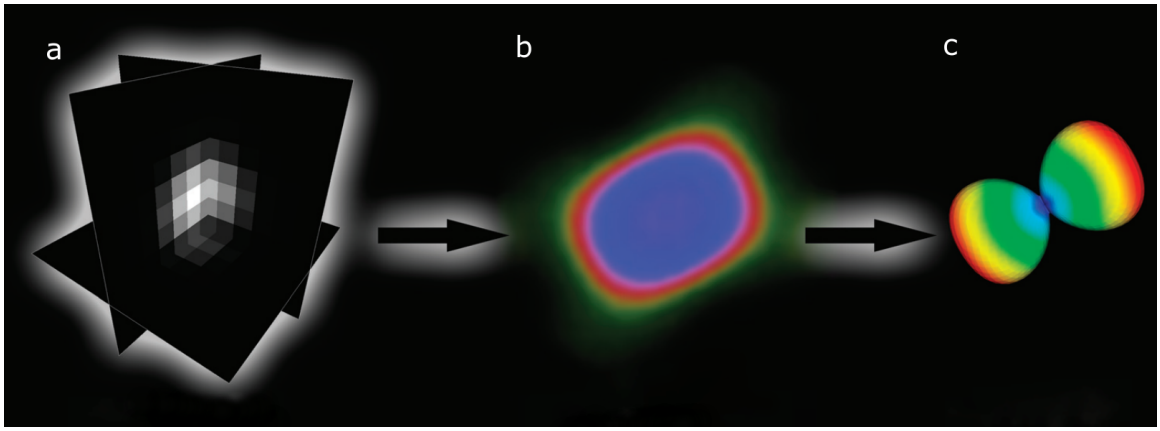

Figure 3.10 – Relation entre le signal de diffusion brut, la fonction de diffusion et l'ODF de diffusion. En a) Le signal brut $S(q)$. En b) La fonction de diffusion $P(r)$. En c) La fonction de distribution d'orientation de diffusion $\psi(\theta, \phi)$. Adapté de [36].

fréquent de représenter l'atténuation du signal de diffusion $E(q,t)$ comme étant

$$E(q,t) = \frac{S(q,t)}{S_0}, \tag{3.7}$$

avec la relation $q = \frac{g}{|g|}$ et t le temps de diffusion. À partir de cette définition, nous pouvons introduire la fonction de diffusion $P(r)$ représentant la probabilité d'un déplacement relatif r des *spins* durant le temps de diffusion. Cette fonction de diffusion est liée à l'atténuation du signal à travers la transformée de Fourier $\mathcal{F}$ tel que

$$P(r) = \mathcal{F}^{-1}[E(q)]. \tag{3.8}$$

La fonction de diffusion est un outil très puissant, mais il serait intéressant d'ajouter une composante orientationelle à celle-ci. Introduisons maintenant la fonction de distribution d'orientations de diffusion (*diffusion orientation distribution function*, diffusion ODF, dODF) définie par

$$\psi(\theta, \phi) = \int_0^\infty P(r, \theta, \phi) r^2 \mathrm{d}r, \tag{3.9}$$

où θ, ϕ sont des coordonnées sphériques. L'ODF de diffusion est une fonction discrète

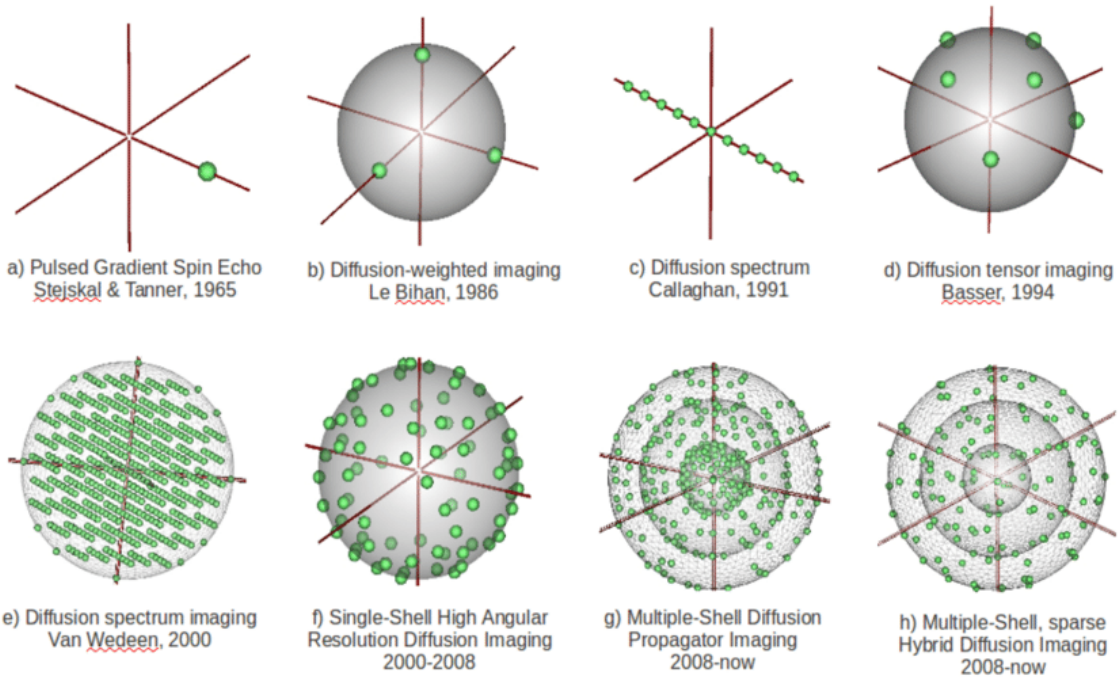

Figure 3.11 – Différents exemples de stratégies d'échantillonnage de l'espace Q. Nous pouvons voir en b) les gradients en x, y, z pour l'imagerie pondérée par la diffusion, en c) les six directions minimum requises pour l'imagerie par tenseurs de diffusion et en f) l'acquisition sphérique requise pour l'imagerie Q-Ball. Tiré de [39].

sur la sphère permettant d'associer une valeur de diffusion à chaque coordonnées sphérique. La figure 3.10 permet d'illustrer le lien entre $S(q)$, $P(r)$ et $\psi(\theta, \phi)$. Afin d'obtenir celle-ci, plusieurs stratégies sont possibles : échantillonner le plus complètement possible le volume de la sphère (*Diffusion Sprectrum Imaging* [90, 130]), échantillonner les sommets d'une sphère discrétisée dans l'espace q (*Single-Shell HARDI* [70, 60, 117]), ou plusieurs sphères de rayons différents (*Multiple-Shell HARDI* [34, 2]). La figure 3.11 illustre les différentes stratégies d'échantillonage de l'espace q.

Afin d'offrir une représentation compacte du signal de diffusion S sur la sphère, les harmoniques sphériques [37, 62, 3, 30] ont été proposées.

Les harmoniques sphériques (*spherical harmonics*, SH) sont l'équivalent de la transformée de Fourier, mais sur la sphère. Elles permettent d'approximer n'importe quelle fonction arbitraire sur la sphère en formant une combinaison linéaire de fonctions

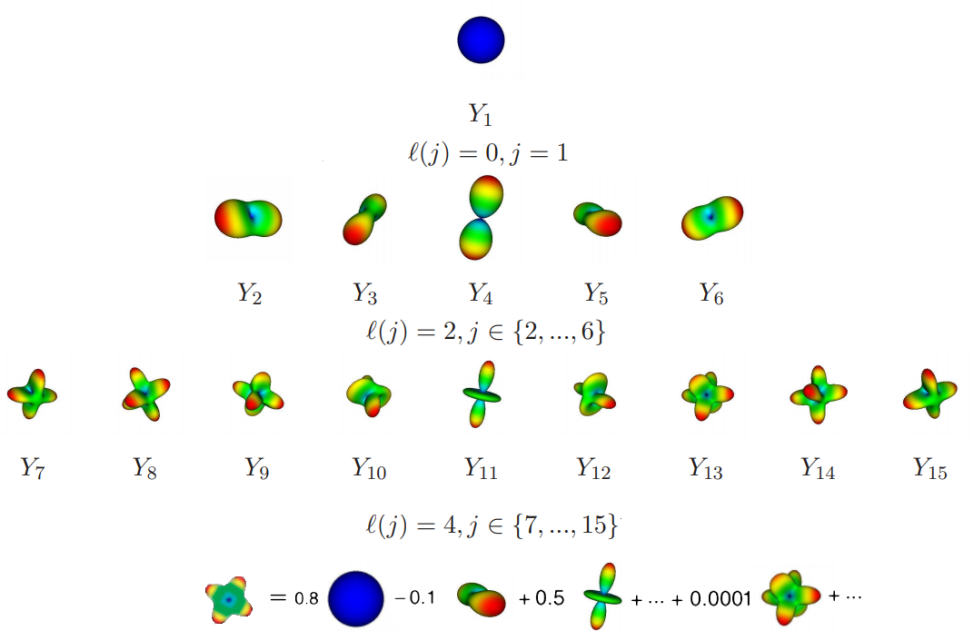

Figure 3.12 – Illustration des fonctions de base d'harmoniques sphériques modifiées d'ordre 0, 2 et 4, ainsi que de leur combinaison afin de former une fonction arbitraire. Adapté de [36].

de bases plus simples [30].

Les fonctions de bases sur la sphère peuvent être définies comme

$$Y_l^m(\theta, \phi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} P_m^l(\cos(\theta) e^{im\phi}), \tag{3.10}$$

où P_l^m est un polynôme de Legendre, l et m sont respectivement l'ordre $l \in N_0$ et la phase m t.q. $-l \leq m \leq l$ associés à la fonction de base. Les harmoniques sphériques permettent de représenter n'importe quelle fonction sur la sphère, mais puisque le signal de diffusion est supposé symmétrique et réel, permettons-nous d'apporter les modifications suivantes : seules les bases paires $k = 0, 2, 4..l$ seront considérées, un nouvel index $j(k, m) = \frac{(k^2+k+2)}{2} + m$ sera défini ainsi qu'une nouvelle famille de

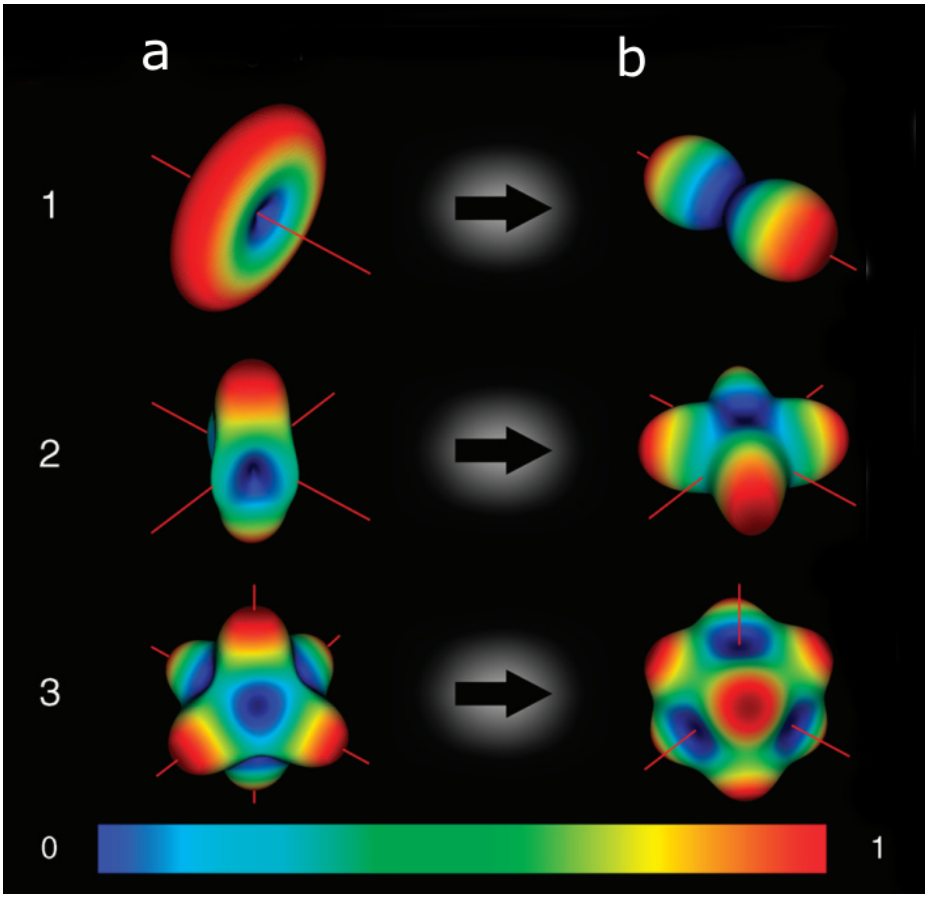

Figure 3.13 – Représentation du signal HARDI brut et l'ADC reconstruite pour plusieurs populations de fibres. En a) Le signal HARDI brut. En b) L'ADC équivalente. Adapté de [37].

fonctions de bases Y_j t.q.

$$Y_j(\theta,\phi) = \begin{cases} \sqrt{2}Re(Y_k^m(\theta,\phi)) & \text{si } -k \leq m < 0 \\ Y_k^0(\theta,\phi) & \text{si } m = 0 \\ \sqrt{2}Img(Y_k^m(\theta,\phi)) & \text{si } 0 < m \leq k \end{cases} \tag{3.11}$$

avec $Re(Y_k^m)$ et $Img(Y_k^m)$ respectivement les parties réelles et imaginaires de Y_l^m. La figure 3.12 permet d'illustrer ces nouvelles fonctions de bases.

Grâce à cette définition, nous pouvons, par exemple, reconstruire le signal de

diffusion HARDI par une combinaison linéaire de la nouvelle base Y_j tel que

$$S(\theta, \phi) = \sum_{j=1}^{R} c_j Y_j(\theta, \phi), \tag{3.12}$$

avec $R = \frac{(l+1)(l+2)}{2}$ le nombre de termes de la base modifiée. Par contre, nous devons obtenir les coefficients c_j. En posant $\mathbf{S}$ le vecteur du signal de diffusion $N \times 1$ (avec N le nombre de directions d'acquisitions), $\mathbf{C}$ le vecteur de coefficients de SH $R \times 1$ et $\mathbf{B}$ la matrice $N \times R$ de bases de SH en tant que :

$$\mathbf{B} = \begin{pmatrix} Y_1(\theta_1, \phi_1) & \dots & Y_R(\theta_1, \phi_1) \\ \vdots & \ddots & \vdots \\ Y_1(\theta_N, \phi_N) & \dots & Y_R(\theta_N, \phi_N) \end{pmatrix}, \tag{3.13}$$

de telle sorte que $\mathbf{S} = \mathbf{CB}$, nous pouvons obtenir les coefficients de SH avec la méthodes des moindres carrés suivante :

$$\mathbf{C} = (\mathbf{B}^T\mathbf{B} + \lambda\mathbf{L})^{-1}\mathbf{B}^T\mathbf{S}, \tag{3.14}$$

où $\lambda\mathbf{L}$ sont un terme et une matrice de régularisation [30].

De retour au signal de diffusion : celui-ci, ainsi que l'ADC correspondante tel que présentés aux équations 3.2 et 3.3 peuvent être reformulés afin d'utiliser les multiples directions des méthodes HARDI selon :

$$S(b)_{g_i} = S_0 e^{(-bD(g_i))} \tag{3.15}$$

$$D(g_i) = \frac{-1}{b} \log\left(\frac{S(b)_{g_i}}{S_0}\right) \tag{3.16}$$

et $g_i,\ i \in 1..N$ les N directions de gradients acquises. L'ADC peut ensuite être représentée par des SH et être visualisée sur la sphère, tel que l'indique la figure 3.13. Par contre, on peut observer que l'ADC tend à présenter des valeurs élevées dans des directions orthogonales aux fibres sous-jacentes. L'ADC ne permet donc pas de bien représenter les populations de fibres sous-jacentes, et un autre représentation sera

nécessaire.

L'imagerie par Q-Ball [143] est basée sur l'utilisation de la transformée de Funk-Radon (TFR) afin d'estimer l'ODF à partir du signal de diffusion. Grossièrement, la TFR est la somme sur une sphère de l'équateur orthogonal à un vecteur donné x. De plus, la TFR donne une bonne estimation de l'ODF tel que

$$\psi(\theta, \phi) \approx TFR_{q'}[E(q)], \tag{3.17}$$

où q' est le rayon de la sphère d'échantillonage.

La TRF requiert par contre l'intégration de plusieurs grands cercles sur la sphère. Afin de résoudre ces intégrales [143], plusieurs méthodes ont été proposées, mais nous nous tournerons vers

Dans la même veine, la TFR peut être définie pour un vecteur unitaire de direction u par notre série modifiées de SH tel que

$$\psi(u) = \sum_{j=1}^{R} 2\pi P_{l_j}(0) c_j Y_j(u), \tag{3.18}$$

où $P_l(0)$ est le polynôme de Legendre de degré l évalué à 0 tel que

$$P_{l_j}(0) = \begin{cases} 0 & \text{si } l \text{ est impair.} \\ (-1)^{l_j/2} \frac{1 \cdot 3 \cdot 5 \ldots (l_j - 1)}{1 \cdot 3 \cdot 5 \ldots l_j} & \text{si } l \text{ est pair.} \end{cases} \tag{3.19}$$

L'équation 3.18 peut aussi être résolue par la méthode des moindres carrés de façon similaire à l'équation 3.13 tel que

$$\mathbf{C} = \mathbf{P}(\mathbf{B}^T\mathbf{B} + \lambda\mathbf{L})^{-1}\mathbf{B}^T\mathbf{S}, \tag{3.20}$$

avec $\mathbf{P}$ la matrice $R \times R$ des polynômes de Legendre de degré 0 de l'équation 3.18 [30].

3.1.6 Déconvolution sphérique contrainte

L'imagerie par Q-Ball permet de mieux représenter le signal de diffusion que l'imagerie par tenseur de diffusion. Les populations de fibres sont mieux représentées,

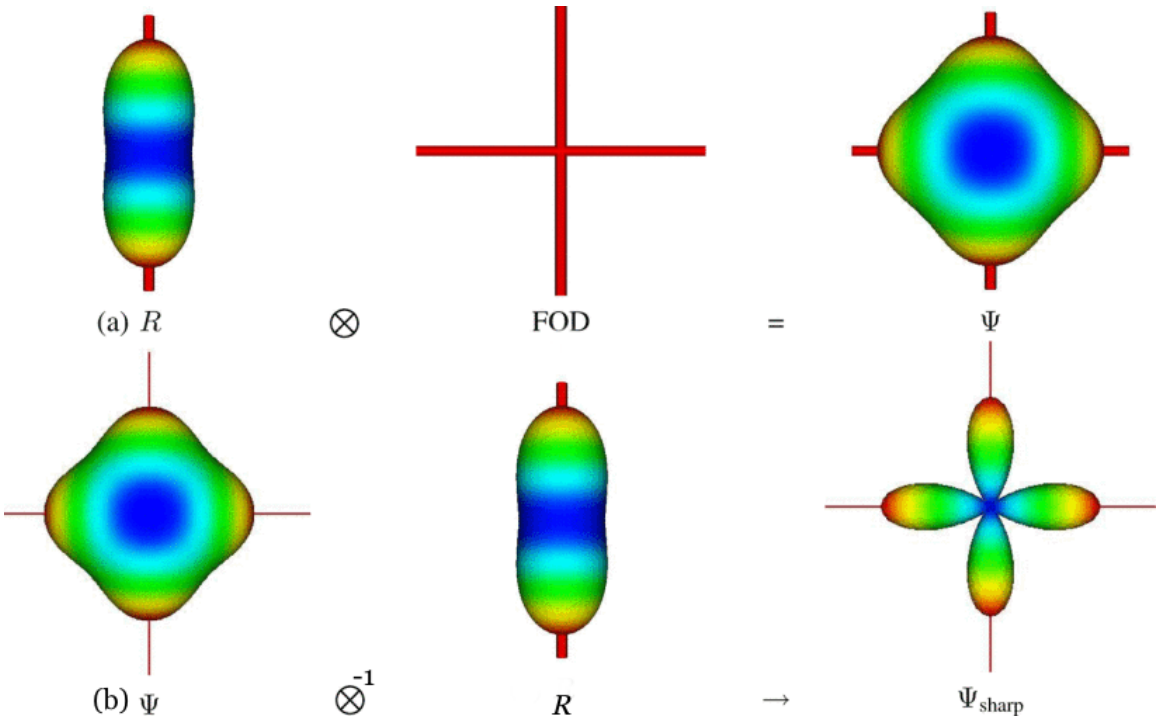

Figure 3.14 – Illustration de "l'aiguisage" de l'ODF de diffusion par la déconvolution sphérique, où en (a) l'ODF de diffusion est présumé être une version "floue" de la distribution des fibres de la matière blanche. Les croisements de fibres sont clairement mieux représentés suite à la déconvolution, en (b). Adapté de [32].

et la diffusion n'est pas supposée uniforme dans le voxel, mais devient plutôt représentée par une fonction arbitraire sur la sphère, permettant à celle-ci de prendre (presque) n'importe qu'elle forme. Par contre, l'ODF de diffusion demeure "floue" ; après tout, celle-ci représente le mouvement des molécules d'eau, qui n'est pas exclusivement parallèle aux fibres. Tel que représentés dans la figure 3.14, les croisements de fibres demeurent trop "lisses", où la probabilité de diffusion demeure élevée entre les fibres plutôt que le long de celles-ci.

La déconvolution sphérique [30] a été proposée comme outil pour palier à ce problème. Originellement proposée sur le signal HARDI brut [135], celle-ci a été adaptée en présumant que l'ODF de diffusion est le résultat de la "convolution" entre la vraie distribution de fibres et un "noyau" d'ODF de diffusion. Pour obtenir la vrai distribution de fibres sous-jacente, l'opération de déconvolution, soit l'inverse de la

convolution, peut être appliquée à l'ODF de diffusion.

La transformée de déconvolution sphérique (*spherical deconvolution transform*, SDT) a été proposée pour effectuer l'opération de déconvolution. À partir d'un ODF de diffusion R provenant d'une seule population de fibres, nous pouvons obtenir l'ODF "aiguisé" ψ_{sharp} selon

$$\psi_{\text{sharp}}(\mathbf{u})_p = \sum_{j=1}^{R} 2\pi P_{\ell_j}(0) c_{j_{\text{sharp}}} Y_j(\mathbf{u}). \tag{3.21}$$

où

$$c_{j_{\text{sharp}}} = \frac{c_j}{f_j}, \quad \text{avec} \quad f_j = 2\pi \int_{-1}^{1} P_{\ell_j}(t) R(t) dt. \tag{3.22}$$

L'intégrale sur $P_{\ell_j}(t)R(t)$ peut être calculée grâce au théorème de Funk-Hecke [32] et l'équation 3.21 de la même façon que l'équation 3.18. L'ODF résultante sera appelée ODF de fibres (*fiber ODF*, fODF). La *déconvolution sphérique contrainte* (Constrained Spherical Deconvolution, CSD) permet d'ajouter une contrainte de non-négativité sur les élément de la FOD, éliminant les régions négatives physiquement impossibles [131]. À noter que la

3.2 Tractographie

La tractographie est un outil très puissant permettant d'investiguer la structure de la matière blanche du cerveau humain. Celle-ci a permis d'obtenir des résultats quantitatifs par faisceaux lors de l'étude de maladies telles que la maladie d'Alzheimer, la schizophrénie, les accidents vasculaires cérébraux et l'épilepsie [72] en plus de s'imposer comme outils de choix pour la connectomique, soit l'étude du réseau complexe qu'est le cerveau [20].

Maintenant que nous savons comment représenter le signal de diffusion, que faire avec ? Revenons au vif du sujet de ce chapitre : la tractographie tente d'inférer la structure de la matière blanche grâce à l'information de diffusion. Une façon de représenter la matière blanche serait de voir les fibres comme des courbes continues dans un espace tri-dimensionnel dont nous tenterons d'estimer la forme. À défaut d'avoir la définition exacte de ces courbes, nous devrons trouver une façon de les

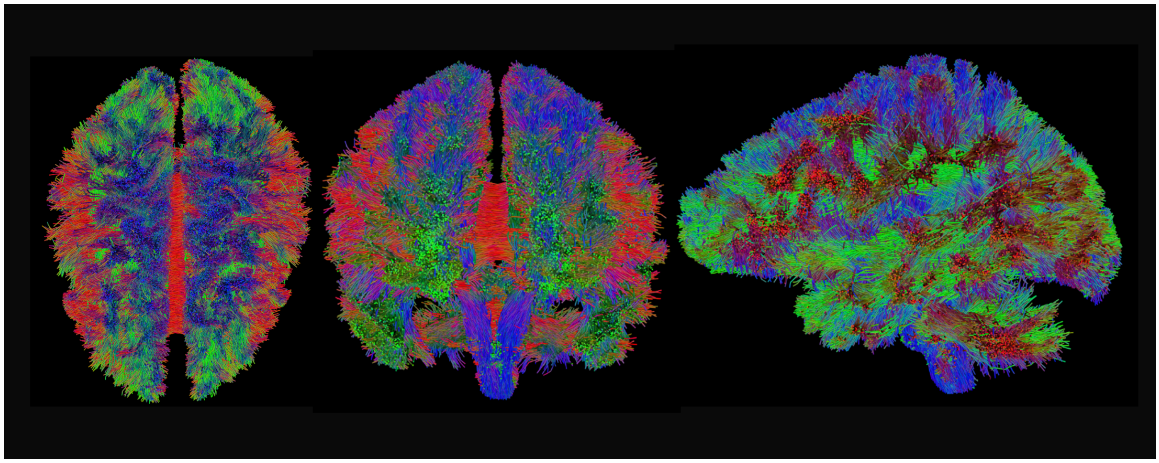

Figure 3.15 – Tractogramme en vue axiale, coronale et sagitale, les couleurs représentant les directions locales.

estimer grâce à l'information de diffusion et plus particulièrement du champ d'ODF reconstruit à la dernière section.

Dans cette section, nous présenterons comment se servir de ce champs d'ODF afin de produire un tractogramme, soit une représentation virtuelle de la structure de la matière blanche. Nous commencerons par présenter un algorithme général de tractographie ainsi que quelques choix disponibles quant à l'implémentation de celui-ci. Nous présenterons aussi quelques améliorations apportées à l'algorithme de base, ainsi que les problèmes connus affligeant ceux-ci. Finalement, nous terminerons avec un aperçu des outils de validation des algorithmes de tractographie ainsi qu'un survol de l'état de l'art quant à la tractographie par apprentissage supervisé.

3.2.1 Algorithme général

La méthode d'Euler est une façon de résoudre par approximation des équations différentielles en ayant une valeur initiale. Si on visualise l'équation différentielle à résoudre comme une courbe où il est possible d'obtenir la tangente en tout point tel que

$$\frac{ds_t}{dt} = v_t, \tag{3.23}$$

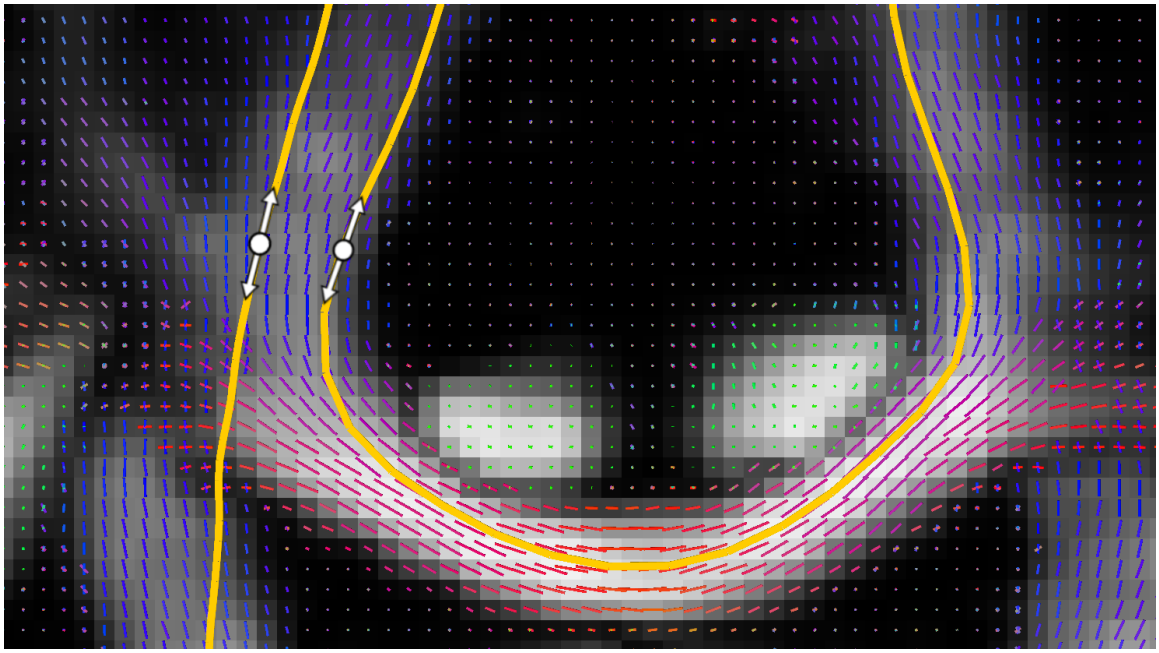

Figure 3.16 – Visualisation du processus de tractographie via deux tracts reconstruites. Les petits "bâtons" représentent le pic d'orientation à chaque voxel. Les points blancs représentent deux seeds choisis pour initialiser le tracking. Les flèches représentent le tracking "avant" et "arrière", et les lignes jaunes représentent les tracts reconstruites.

avec v_t la tangente à s_t au temps t, et en supposant un point initial p_0, la méthode d'Euler dicte qu'on peut approximer s en se "promenant" sur celle-ci selon

$$p_{t+1} = p_t + v_t \Delta s, \tag{3.24}$$

avec Δs un "pas de temps" raisonnablement petit. Tel que mentionné précédemment, la matière blanche peut être représentée comme un ensemble de courbes que nous aimerions approximer. La méthode d'Euler devient alors particulièrement attrayante : en choisissant un endroit de départ dans celle-ci, nous pouvons obtenir la tangente en tout point dans la matière blanche grâce aux maximums locaux, les "pics" (*peaks*) des fODFs reconstruits, et suivre ces tangentes jusqu'à atteindre la matière grise. Ce principe se nomme la *tractographie* [18].

Plus formellement, le processus de *tracking* requiert un champ de vecteurs $\mathbf{v}$ représentant la ou les directions supposées des fibres de la matière blanche à chaque

voxel, un masque $\mathbf{m}_{tracking}$ indiquant où il est possible de *tracker*, et un point 3D de départ p_0. Nous tenterons de trouver tous les points $p_t \in s$ d'une *tract* (ou *streamline*) s en faisant des pas successifs dans le champs $\mathbf{v}$ tel que

$$p_{t+1} = p_t + \mathbf{v}(p_t)\Delta s, \tag{3.25}$$

où Δs est un pas assez petit pour obtenir une précision intra-voxel. Le *tracking* se poursuit jusqu'à ce qu'un critère d'arrêt soit atteint : le point p_t est hors du masque de tracking ou l'angle (en radians) entre les deux derniers segments de la tract reconstruite, calculé selon

$$\theta = \frac{s_t \cdot s_{t-1}}{||s_t||||s_{t-1}||} \quad \text{avec} \quad s_t = p_t - p_{t-1}, \tag{3.26}$$

est trop grand. L'encadré 3.1 décrit un algorithme de tractographie simpliste, où la fonction "*obtenirDirection*" s'occupe de retourner le pic d'ODF ayant l'orientation la plus semblable au dernier segment s_t de la tract tel que

$$v_t = \max_v s_t \cdot v \; \forall \; v \; \in \; \mathbf{v}(p_t), \tag{3.27}$$

ainsi que d'inverser la direction des peaks au besoin afin que

$$s_t \cdot v \geq 0 \; \forall \; v \; \in \; \mathbf{v}(p_t). \tag{3.28}$$

Puisque la direction des vecteurs tangents est indéterminée (rappelons-nous que la diffusion est supposée symétrique), la direction prise lors du tracking est inconnue et aussi valable que la direction inverse. C'est pourquoi le tracking doit s'effectuer par "l'avant" et "l'arrière", où chaque pas de tracking avant est conséquent avec la direction choisie du vecteur tangent et chaque pas de tracking "arrière" est conséquent avec la direction inverse.

Une pléthore d'algorithmes a été développée depuis les début de la tractographie, et l'algorithme général présenté dans l'encadré 3.1 n'est qu'une version simpliste d'un tel algorithme.

Quelques détails d'implémentation ont aussi été cachés dans le but de simplifier l'algorithme, comme le calcul de la prochaine direction, par "plus-proche voisin" ou

Algorithm 3.1 Tractographie déterministe

$P_0 \leftarrow \{p_0^i \; \forall \; i..N\}$ sélectionnés dans le masque $\mathbf{m}_{\text{points initiaux}}$ ▷ Points initiaux
$S = \{\}$ ▷ Banque de tracts
procedure Tracking($\theta_{\max}, \Delta s, \mathbf{m}_{\text{tracking}}, \mathbf{v}$)
    for $p_0^i \in P$ **do**
        $t \leftarrow 0$
        while $p^i_{t_{avant}} \in \mathbf{m}_{tracking}$ et $\theta < \theta_{\max}$ **do** ▷ Tracking vers *l'avant*
            $p^i_{t_{avant}+1} = p^i_{t_{avant}} + obtenirDirection(\mathbf{v}(p_t^i), \Delta s)$
            $t_{avant} \leftarrow t_{avant} + 1$
        $t \leftarrow 0$ ▷ Le tracking est réinitialisé vers l'arrière
        while $p^i_{t_{arriere}} \in \mathbf{m}_{tracking}$ et $\theta < \theta_{\max}$ **do** ▷ Tracking vers *l'arrière*
            $p^i_{t_{arriere}+1} = p^i_{t_{arriere}} - obtenirDirection(\mathbf{v}(p_t^i), \Delta s)$
            $t \leftarrow t + 1$
        $S \leftarrow S \cup (p^i_{0:t_{avant}} + p^i_{0:t_{arriere}})$
    return S

par interpolation des peaks avoisinant, par exemple. Les étapes de tracking "avant" et "arrière" peuvent aussi être effectuées en parallèle, ainsi que le tracking sur plus d'un seed à la fois.

De plus, la méthode d'Euler n'est qu'une forme d'approximation des streamlines, mais des méthodes d'ordre supérieur peuvent aussi être employées [18]. Dans les prochaines sous-sections, certains détails pertinents pour le chapitre 4 seront abordés. Des revues plus exhaustives [57, 69, 98] sont disponibles au lecteur souhaitant en apprendre plus.

3.2.2 Stratégies d'initialisation

Tel que mentionné dans l'algorithme 3.1, un masque $\mathbf{m}_{\text{points initiaux}}$ doit être fourni à l'algorithme afin de définir l'emplacement des points initiaux (*seeds*). Trois grandes stratégies d'initialisations des points initiaux sont généralement considérées.

Une région d'intérêt peut être sélectionnée afin d'initialiser les fibres à partir d'un endroit spécifique. Par exemple, dans le but de reconstruire son faisceau préféré, une région d'intérêt pourrait être sélectionnée au centre ou aux extrémités de celui-ci. Il sera par contre plus difficile d'espérer reconstruire l'entièreté de la matière blanche à partir d'une seule région.

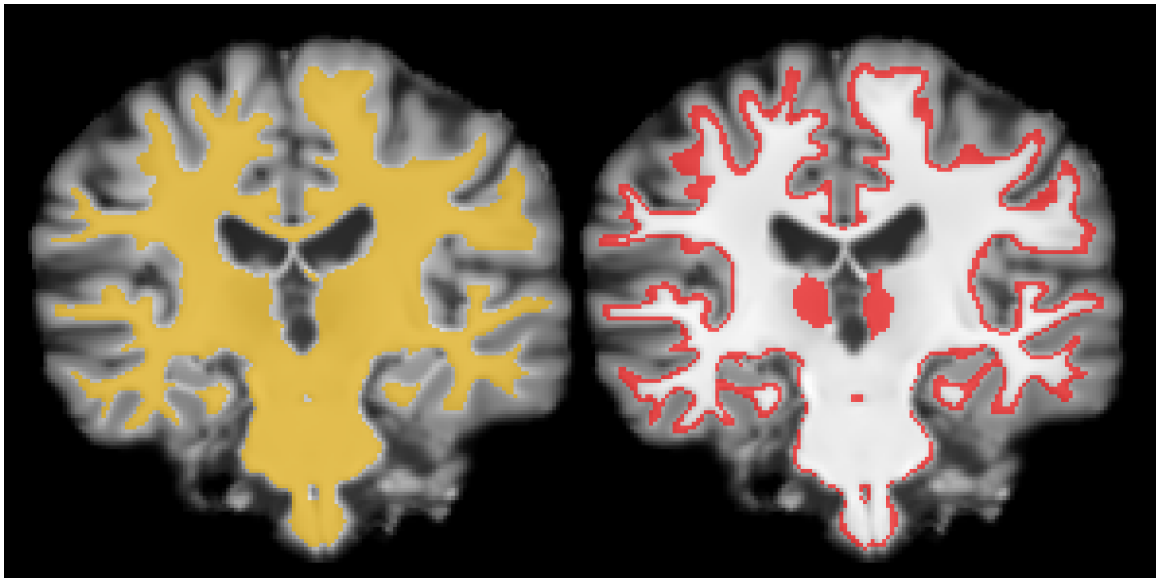

Figure 3.17 – Visualisation des masques de *seeding*. À gauche, le masque couvre la matière blanche, alors qu'à droite, il couvre la frontière matière-blanche/matière-grise.

Si un tractogramme représentant l'entièreté de la matière blanche est désiré, deux stratégies s'offrent à l'utilisateur : la première consiste à sélectionner les voxels à la frontière entre la matière grise et la matière blanche. Cette stratégie est intuitive en se rappelant que les axones commencent et s'arrêtent dans la matière grise et voyagent dans la matière blanche. Il serait donc logique de commencer les tracts à l'endroit où les axones débutent. Par contre, la tractographie n'étant pas un processus parfait, les tracts résultantes ne sont pas garanties de couvrir tout le volume de matière blanche, encore moins de se rendre à destination [69].

L'alternative est de placer des seeds uniformément dans le volume de matière blanche. Ceci a pour avantage d'assurer une plus grande couverture du volume de la matière blanche, puisque des tracts en émaneront en tous points. Par contre, certains faisceaux peuvent se retrouver *sur-définis*, puisqu'ils seront reconstruits à partir de bien plus de points [69]. La figure 3.17 illustre les deux options.

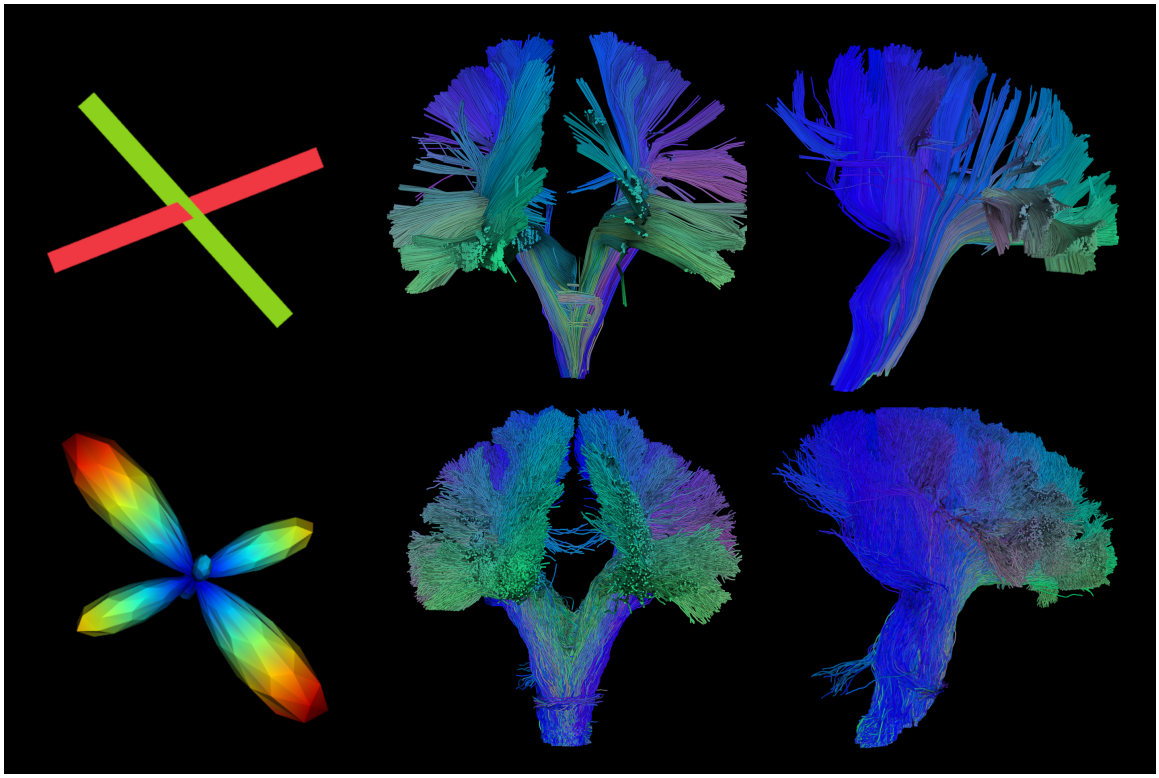

Figure 3.18 – Visualisation des composantes de tracking déterministes et probabilistes. Rangée du haut : peaks suivis par un algorithme déterministe, ainsi que les faisceaux cortico-spinaux, frontopontiques et parieto-occipitaux-pontiques reconstruits par un tracking déterministe et extraits par RecoBundlesX [52]. Rangée du bas : glyph illustrant une fODF servant de fonction de masse pour la direction à prendre par un tracking probabiliste, ainsi que les même faisceaux, mais reconstruits par un algorithme probabiliste.

3.2.3 Tracking déterministe vs. probabiliste

Le tracking tel que présenté dans l'encadré 3.1 présume une modélisation parfaite des fibres sous-jacentes, et donc d'une directionalité du signal sans bruit. Par contre, l'estimation des fODFs est loin d'être un processus parfait : l'acquisition de l'image de diffusion produit une image bruitée et la modélisation du signal n'est pas parfaite. Un algorithme de tractographie peut alors ne reconstruire que les faisceaux les plus "collés" sur le signal, sans "explorer" les faisceaux moins probables [134].

Dans le but de prendre en compte cette accumulation de bruit, les algorithmes de

tractographie *probabilistes* (par opposition aux algorithmes *déterministes* présentés jusqu'à maintenant) proposent de propager la tract, non pas selon la ou les directions principales, mais selon la fODF sous-jacente en l'utilisant comme fonction de masse (tout en respectant les contraintes d'arrêt de tracking) sur la direction à prendre.

Tel que l'indique la figure 3.18, les algorithmes probabilistes permettent de mieux "remplir" le volume de tracking, produisant un tractogramme plus plausible anatomiquement et permettent d'explorer des chemins que les algorithmes déterministes ne prendraient pas. Par contre, les algorithmes probabilistes peuvent "trop" remplir le volume de tractographie et, de par leur nature exploratoire, aller jusqu'à produire des tracts connectant deux régions qui ne devraient pas l'être [134]. Malgré cela, les algorithmes de tractographie modernes tendent tous à être probabilistes [134].

3.2.4 Problèmes liés à la tractographie

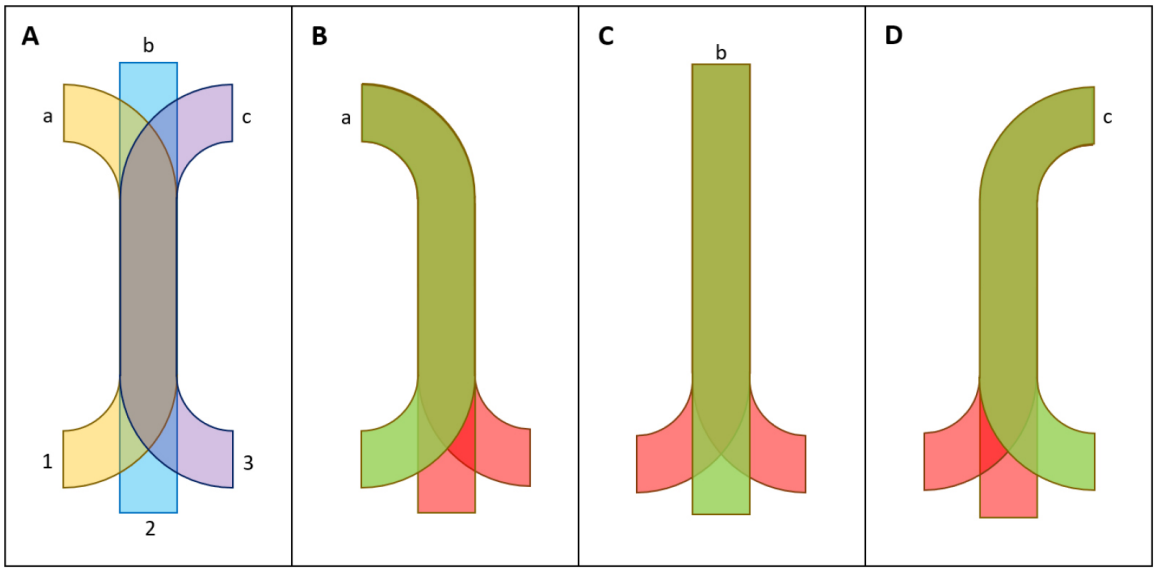

Figure 3.19 – Visualisation de l'effet de *bottleneck*, où trois faisceaux traversent le même goulot d'étranglement. Du point de vue d'un algorithme de tractographie, les sorties $1, 2, 3$ sont toutes valides localement lorsqu'il se trouve à la fin du goulot, au croisement proche de a, b, c. L'algorithme a donc plus de chance de reconstruire un faisceau *faux-positif* que *vrai-positif*. Tiré de [114].

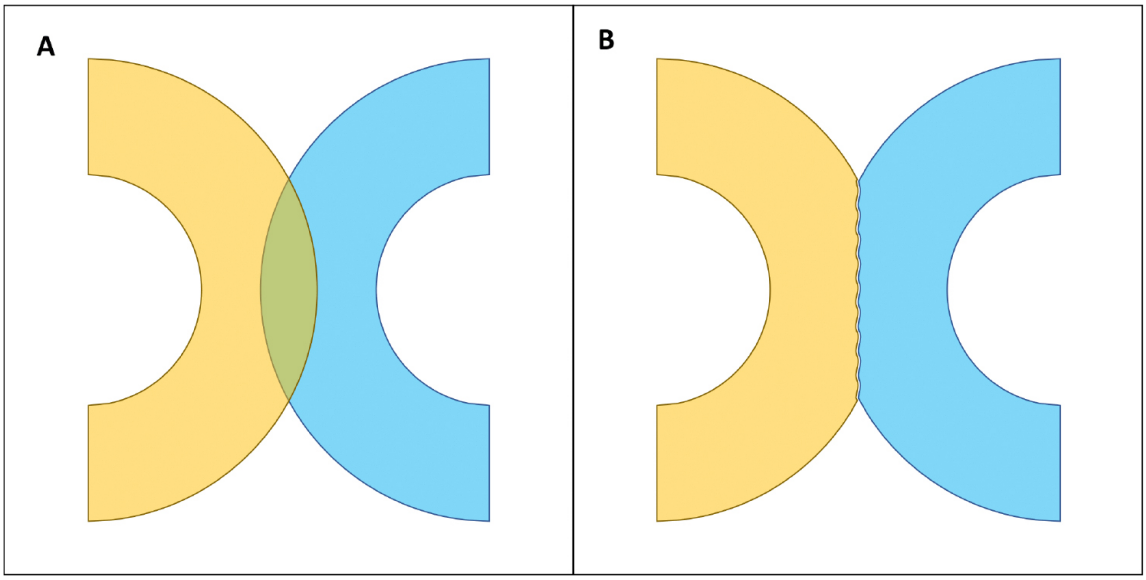

Figure 3.20 – Illustration de l'effet de mur. Le signal au croisement des deux faisceaux n'est pas assez précis pour bien représenter l'intersection et se retrouve moyenné. Les tracts passant par cet endroit tendent à rester de "leur côté" du signal plutôt que de bien prendre le virage. Tiré de [114].

Depuis ses débuts, l'engouement envers la tractographie ne cesse de grandir. Par exemple, le nombre d'articles publiés par années ayant le mot-clé "*Tractography*" sur le répertoire PubMed Central a plus que doublé ces dix dernières années, pour dépasser le cap des 800 en 2018 [114]. Par contre, la tractographie n'est pas sans problèmes et il est facile d'avoir des mécompréhensions face aux tractogrammes produits par son algorithme de tractographie.

Tel que mentionné précédemment, le masque de points initiaux influence grandement la reconstruction des faisceaux. Utiliser un masque de points initiaux couvrant toute la matière blanche peut mener à une saturation des faisceaux plus longs, introduisant potentiellement un biais lors de la reconstruction du connectome [57, 114]. De plus, initialiser le tracking à partir de toute la matière blanche peut mener à la reconstruction de faux-positifs, des faisceaux anatomiquement improbables [114]. Ceux-ci sont appelés *faisceaux invalides*, (invalid bundles, IB), par opposition aux faisceaux biologiquement probables, les *faisceaux valides*, (valid bundles, VB). À l'inverse, un masque de point initiaux à la frontière WM/GM peut mener certains faisceaux ayant

peu de couverture corticale à être sous-représentés, ou même non-reconstruits.

Les critères d'arrêts choisis peuvent aussi grandement influencer la reconstruction des faisceaux. Par exemple, l'utilisation d'un masque de tracking binaire tel que mentionné précédemment peut mener à un arrêt prématuré du tracking. En effet, un masque de tracking binaire calculé en seuillant la FA ou une image T1, sans prendre en compte les effets de volume partiel, peut générer des chemins sinueux trop durs à reconstruire [57]. De ce fait, les faisceaux plus longs, même initiés à partir de la matière blanche, peuvent être plus difficiles à reconstruire puisque les tracts qui les composent ont plus de chance de rencontrer un obstacle et de s'arrêter prématurément [57, 114].

Dans la même veine, les faisceaux peuvent aussi se "nuire" entre-eux de plusieurs façons. Par exemple, l'effet *goulot* (*bottleneck*), tel qu'illusttré par la figure 3.19, se produit lorsque plusieurs faisceaux doivent passer par le même endroit puis se séparer à nouveau, tel qu'observé dans la capsule interne ou dans le tronc temporel, par exemple [114]. Sans *a priori* sur la direction à prendre, toutes les directions sont valides au sens de la tractographie, menant à beaucoup de faux-positifs [114].

L'effet de mur (*wall-effect*), tel qu'illustré par la figure 3.20, est présent lorsque deux faisceaux se croisent et se chevauchent brièvement pour se séparer à nouveau (*kissing bundles*). Dans bien des cas, le croisement des faisceaux est mal représenté dans le signal, où l'ODF n'est pas aussi aiguisée qu'elle devrait l'être pour bien démêler les deux faisceaux. Les peaks extraits sont donc un mélange des deux faisceaux, imposant aux tracts de se redresser plutôt que de prendre le virage. Visuellement, les tracts reconstruites donnent l'impression d'avoir frappé un mur à l'intersection des deux faisceaux [114].

Dans des régions où les croisement de faisceaux sont bien définis, il est toujours possible qu'un algorithme de tractographie prenne le mauvais virage si l'angle entre les faisceaux est faible. Lors du parcours de la région à l'intersection de deux ou plusieurs faisceaux, une accumulation d'erreurs lors des pas de tracking peut ultimement mener un algorithme à choisir le mauvais croisement, tel qu'observé au croisement du faisceau pyramidal et du faisceau arqué [114].

Les algorithmes de tractographie ne sont pas seulement biaisés dans la densité de reconstruction du corps des faisceaux, mais aussi dans leur terminaison. Le biais gyral

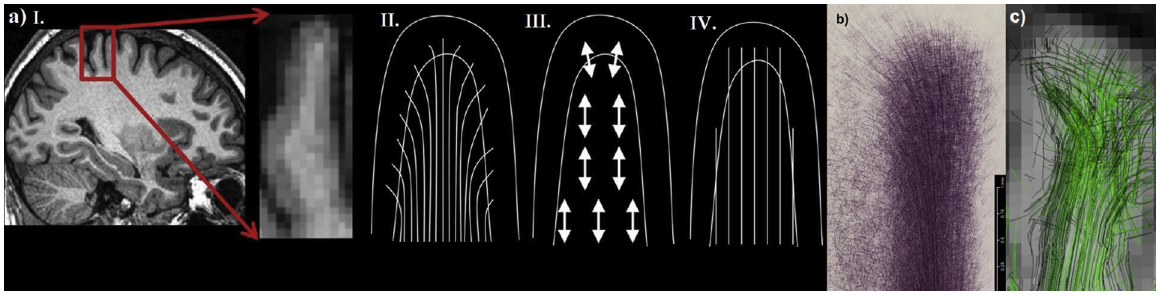

Figure 3.21 – Représentation du biais gyral. En a) I) Zoom sur un replis cortical II) Ce qui est attendu des tracts III) La direction du signal de diffusion mesuré IV) Les tracts reconstruites b) Coloration post-mortem c) Tracts reconstruites. Adapté de [124].

(*gyral bias*) [114, 120] demeure un problème ouvert menant les tracts à se terminer plus souvent dans la couronne (*crown*) des gyrus que les parois (*wall*) ou le "fond" (*fundus*), contrairement à ce que les preuves histologiques indiquent. Ceci peut être expliqué en partie, en plus des limitations de la résolution disponible, par la courbature des fibres se terminant dans les parois ou le fond des gyrus, que les algorithmes ont de la difficulté à reproduire.

Les problèmes mentionnés sont en partie dûs au fait que la tractographie est en soi un problème mal-posé [93] : la tractographie est un processus se servant d'information locale, mais tentant de reconstruite la connectivité globale du cerveau. De ce fait, la tractographie est vouée à reconstruire des faux positifs et d'omettre de vrais-positifs, car les faux positifs correspondent autant au signal local sous-jacent que les vrais positifs.

De plus, chaque voxel du signal de diffusion peut être l'hôte de centaines de milliers de fibres se croisant, chevauchant ou étant sécantes les unes aux autres sans donner lieu à un signal distinguable d'une population de fibres complètement différente. Pire encore, chaque voxel peut être composé de plusieurs tissus, donnant lieu à l'effet de volume partiel. En prenant en compte toutes les embûches possibles, la tractographie ne peut donc que tenter de donner une supposition éclairée quant à la vrai orientation de la matière blanche [69].

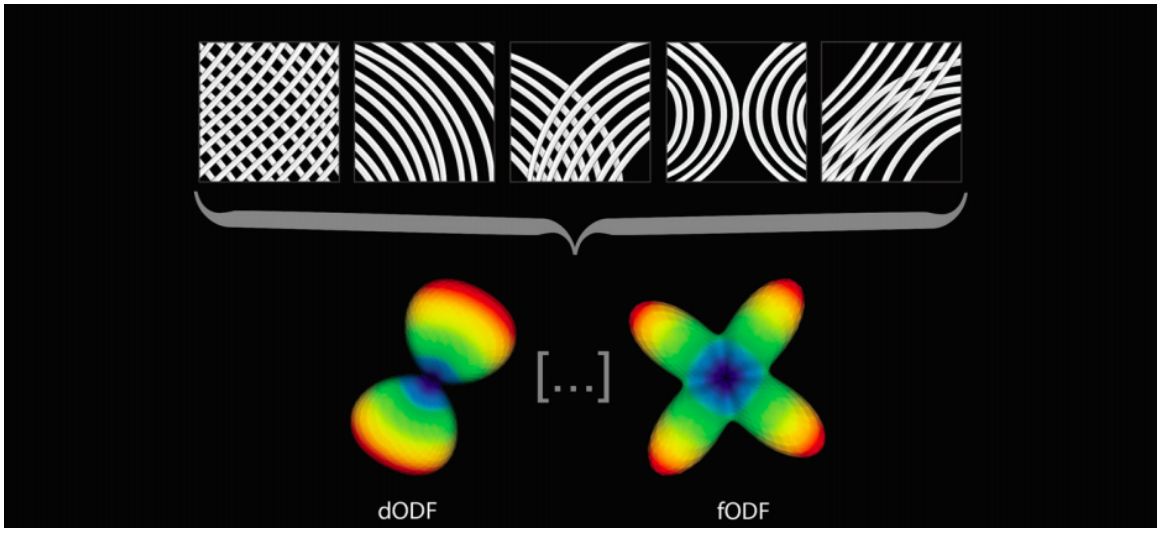

Figure 3.22 – Plusieurs populations très différentes de fibres peuvent mener au même dODF ou fODF reconstruit. Adapté de [37].

3.2.5 Améliorations au processus de tractographie

La dernière sous-section ne devrait pas décourager le lecteur souhaitant se lancer dans la tractographie, mais plutôt l'avertir des pièges potentiels. La tractographie demeure un processus puissant et utile pour mesurer la connectivité du cerveau. Afin d'améliorer les algorithmes de tractographie et de réduire les faux-positifs, il est nécessaire d'injecter des *a priori* anatomiques à l'information traitée. Dans cette sous-section, nous explorerons quelques algorithmes de tractographie *classiques* (non-apprenants) apportant chacun des innovations afin de rendre la tractographie plus fiable.

Particle Filtering Tractography

L'algorithme de *tractographie par filtrage de particules* (Particle Filtering Tractography, PFT) [57] propose, plutôt que d'utiliser un masque de tracking binaire, de prendre en compte les effets de volume partiel à travers deux cartes continues d'inclusion Map^{in} et d'exclusion Map^{ex} de tracts.

Les cartes sont calculées à partir de la segmentation des tissus provenant d'une image pondérée T1 [162] où chaque composante (WM, GM, CSF) se voit attribué sa

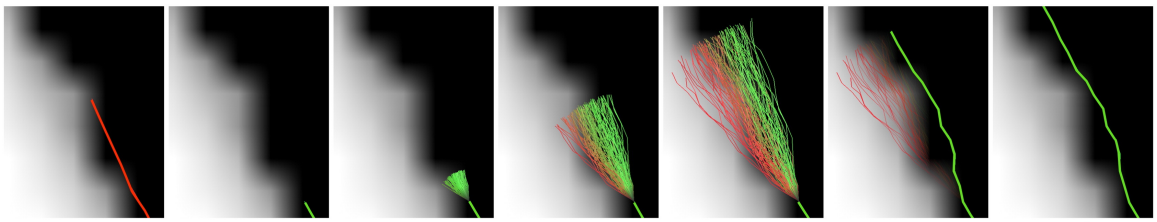

Figure 3.23 – Illustration du processus PFT. a) Une tract s'arrête dans le CSF, en blanc. b) La tract reviens sur ses pas. c,d,e) Les tissus sont échantillonnés, la couleur représente la plausibilité de la trajectoire. f) Une tract est extraite de la distribution d'échantillons. g) Le tracking se poursuit. Tiré de [57]

propre carte. Chaque carte de segmentation contient des valeurs allant de 1 pour le tissu correspondant, 0 pour les autres et des valeurs entre pour les voxels à la frontière. La carte Map^{in} est ensuite composée de la carte de GM et de l'extérieur du masque du cerveau, et la carte de CSF est utilisée en tant que Map^{ex}.

Ces cartes peuvent ensuite être utilisées afin de calculer la probabilité de continuer la propagation P_p^{continue} d'une tract au point p selon

$$P_p^{\text{continue}} = (1 - (Map_p^{in} + Map_p^{ex}))^{\Delta s/\rho}, \tag{3.29}$$

avec ρ la taille des voxels en millimètres. Afin de diminuer le nombre de tracts se terminant ailleurs que dans la matière grise, la probabilité de conserver la tract $P_p^{included}$ à son point p est calculée selon

$$P_p^{included} = Map_p^{in}/(Map_p^{in} + Map_p^{ex}). \tag{3.30}$$

En plus d'utiliser les critères mentionnés plus haut, l'algorithme PFT tente de réduire le nombre de tracts se terminant dans le CSF ou la matière blanche en en effectuant une propagation des tracts basée sur le *filtrage de particules*. Lorsqu'une tract se termine prématurément (soit à l'extérieur de la GM), le processus de tracking revient sur ses pas et le processus de filtrage est enclenché. Le processus de filtrage échantillonne les cartes mentionnées plus haut afin d'attribuer un poids à chaque *particule* générée. Si la particule est valide, celle-ci est propagée vers l'avant. Ce processus est répété pour un nombre d'itérations, résultant en une distribution de

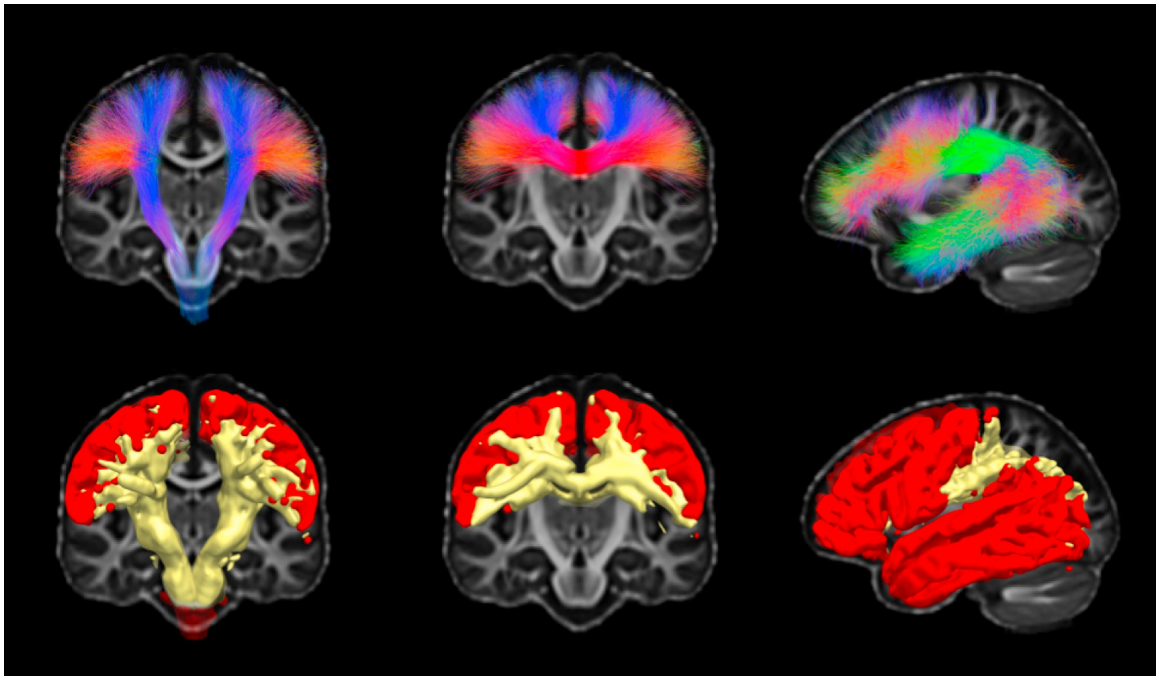

Figure 3.24 – Illustration de l'étape 1 et 2 du prétraitement des données de diffusion dans l'algorithme BST pour le faisceau pyramidal (gauche), le corps calleux (millieu) et le faisceau arqué (droite). La rangée du haut illustre le gabarit de faisceaux, la rangée du bas illustre les masques de tracking et d'initialisation. Tiré de [116].

trajectoires possibles pondérées par les tissus qu'elles traversent. Finalement, une tract est générée à partir de la distribution de trajectoire et le tracking peut se poursuivre si nécessaire.

Bundle-Specific Tractography

Dans le but d'améliorer la couverture des faisceaux sans dépendre d'une sur-initialisation de ceux-ci, ainsi que dans le but de réduire le nombre de faux-positif sans réduire le nombre de vrais positifs, l'algorithme *Tractographie spécifique par faisceaux* (Bundle-Specific Tractography, BST) [116] a été introduit. L'algorithme propose trois phases de prétraitement des données de diffusion afin d'injecter plus d'*a priori* anatomiques dans le processus de tractographie.

La première phase de prétraitement consiste à construire un gabarit du ou des faisceaux à reconstruire : des tractogrammes complets ou partiels provenant de n'im-

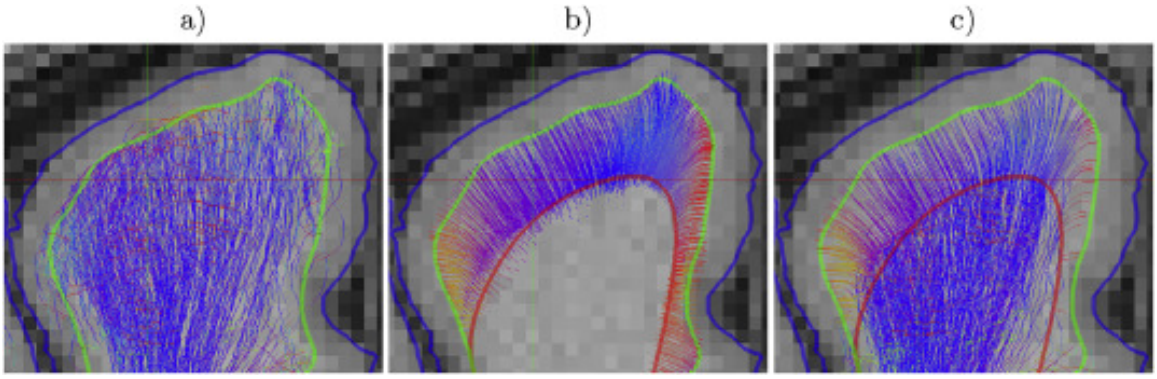

Figure 3.25 – Illustration des avantages de SET. En bleu, la surface de la matière grise. En vert, l'interface WM/GM. En rouge, la surface intermédiaire. a) Tracts générées par un algorithme de tractographie probabiliste. b) Flux de surface c) Un algorithme de tractographie se servant du flux de surface pour initialiser et accueillir les tracts. Tiré de [124].

porte quels algorithmes sont tout d'abord générés, puis segmentés afin d'en retirer les faisceaux d'intérêt. Les faisceaux sont ensuite recalés pour qu'ils existent dans le même espace, puis un filtrage est fait pour tenter d'amenuiser le biais de densité.

Des masques de tracking et d'initialisation sont ensuite extraits des gabarits reconstruits à la première étape, ainsi que les cartes d'inclusion et d'exclusion nécessaires à l'algorithme PFT. Finalement, des *fODF améliorées* (*enchanced FOD*, e-FOD) sont calculées à partir des fODFs extraites de l'image de diffusion et de la *densitié de probabilité de tracts* (*tract orientation distribution*, TOD) [38] afin de réorienter les futures tracts vers les *a priori* reconstruits précédemment.

Finalement, l'algorithme PFT est lancé sur les e-FOD et les cartes d'inclusion et d'exclusion appropriées. Ces améliorations permettent de mieux couvrir l'espace des faisceaux visés, sans compromettre la sensitivité ni la spécificité.

Surface-Enhanced Tractography

Dans le but d'aborder directement le problème du biais gyral, l'algorithme de *tractographie améliorée par surfaces* (Surface-Enhanced Tractography, SET) a été proposé [124]. Afin de rendre moins difficile pour les tracts de prendre la forme d'éventail requise pour bien remplir les plis gyraux, une surface intermédiaire plus

lisse est créée d'où les tracts peuvent partir et s'arrêter. De plus, un "flux de surface" (*surface flow*) est généré entre l'interface GM/WM et la surface intermédiaire. Celui-ci génère les segments initiaux aux tracts se propageant vers le reste de la matière blanche ainsi que les segments finaux des tracts arrivant de la matière blanche vers l'interface GM/WM.

Ces segments de tracts, entre la surface intermédiaire et l'interface GM/WM, sont calculés à partir des normales de la surface corticale. Ceux-ci ne sont donc pas influencés par l'information de diffusion ou les effets de volumes partiels, spécialement présents dans les régions d'interface. Le "flux de surface" est donc bien plus représentatif de ce qui serait attendu des tracts dans les replis gyraux, donnant lieu à des tractogrammes ayant une meilleure couverture corticale, réduisant par le fait même le nombre de tracts invalides par 12% et augmentant la longueur moyenne de celles-ci, en plus de réduire la variabilité et d'augmenter la reproductibilité de la connectivité résultante.

3.2.6 Validation des algorithmes

Tel que mentionné brièvement, une quantité phénoménale d'algorithmes de tractographie est disponible à celui ou celle souhaitant générer des tractogrammes, chacun ayant ses avantages et inconvénients, chacun ayant la possibilité de découvrir des tracts que les autres ne reconstruirons pas. Il est donc difficile pour un néophite (ou même un utilisateur expérimenté) de choisir quel algorithme utiliser selon ses besoins et ses données disponibles.

À ce jour, il persiste un manque de jeux de données permettant de valider la véracité des connections reconstruites par son algorithme de tractographie préféré. Après tout, l'acquisition de tractogrammes dits *vérité terrain* (*ground-truth)* est assez complexe. Comparons-nous à une tâche très populaire : la classification d'images. Plusieurs jeux de données permettant de vérifier les performances d'un algorithme tentant d'accomplir cette tâche [82, 74, 35]. La classification d'images a un aspect très intuitif : les humains sont généralement excellent à reconnaître le contenu d'une image, pouvant mener à des performances de seulement 5.1% d'erreur de classification [111]. Tout en reconnaissant l'immense travail effectué par les concepteurs des jeux de données mentionnés plus tôt, moyennant quelques dollars, il est aujourd'hui possible

de mettre en place un jeu de données très complexe sans trop de soucis à travers la plateforme *Mechanical Turk*[1], par exemple, malgré que cela ne soit pas sans risques [161].

Cependant, générer un jeu de données pour valider la tractographie s'avère généralement plus complexe. Malheureusement, la matière blanche est enfermée dans la boite crânienne et donc difficilement accessible à l'oeil nu. Même si la matière blanche est disponible, elle ne l'est évidemment qu'*ex vivo* et sa dissection est très complexe et destructrice [86]. Des techniques de traçage invasives peuvent être utilisées afin de reconstruire certain faisceaux choisi, mais ces études sont aussi très complexes à effectuer et sont généralement limitées au cerveau du macaque [5].

Sans avoir un accès visuel aux faisceaux de la matière blanche, deux alternatives s'offrent à nous : la première consiste en la conception d'un fantôme, soit un objet simpliste imitant les tissus humains, sur lequel il est facile de valider les faisceaux puisque ceux-ci sont visibles et tangibles. L'alternative est de générer des tractogrammes les plus complets possibles, puis de demander à un ou un groupe d'experts de les nettoyer et segmenter, avec ou sans aide d'un algorithme de segmentation de faisceaux automatique.

Cette technique ne correspond pas tout à fait à la définition d'une vérité terrain comme elle le serait employée dans d'autres domaines comme l'apprentissage supervisé car, même pour un neuroanatomiste chevronné, il demeure impossible d'affirmer l'existence d'une tract qu'à partir d'une image de diffusion. Le terme "vérité terrain" est plutôt réservé à des jeux de données synthétiques ou les fantômes décrits plus haut. De plus, les protocoles de dissection des faisceaux virtuels demeurent quasi-inexistants et la reproductibilité de celle-ci demeure sous-optimale [110, 96]. Néanmoins, le terme "vérité terrain" pour décrire de tels jeux de données est généralement accepté dans la communauté [106] et servira à décrire la sélection non-exhaustive présentée dans cette sous-section. Pour une revue plus complète des jeux de données disponibles pour valider un algorithme de tractographie, voir Poulin et. al [106] ainsi que Schilling et al. [119].

1. https ://www.mturk.com/

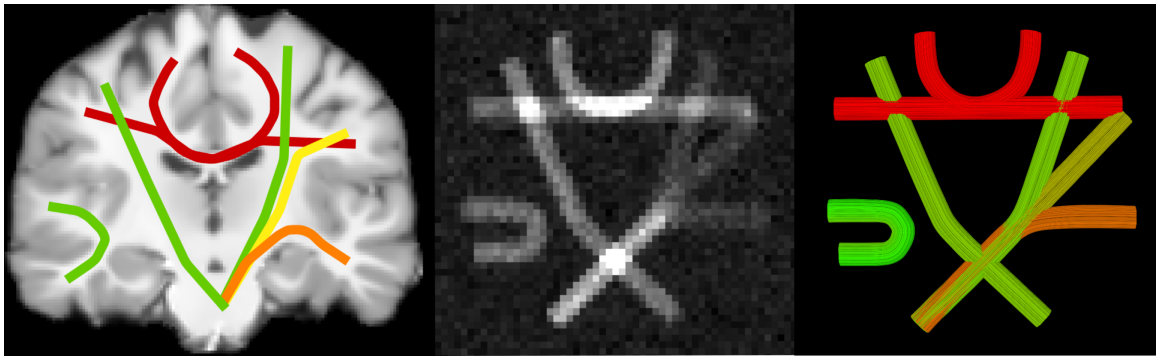

Figure 3.26 – Illustration de composantes du FiberCup. Gauche) Coupe coronale servant d'inspiration au FiberCup. Centre) Signal de diffusion du FiberCup. Droite) Faisceaux vérité-terrain.

FiberCup

Dans le but de valider les techniques HARDI ainsi que les algorithmes de tractographie, le fantôme FiberCup (c.f. fig. 3.26) a été conçu et utilisé dans le concours du même nom en conjonction avec l'édition 2009 de MICCAI [107, 108, 46]. Le FiberCup, composé de fibres d'acryliques entrelacées sur un plan presque-2D, est conçu pour imiter une tranche coronale d'un vrai cerveau humain. Puisque le fantôme a été fabriqué à la main et possède une géométrie relativement simple, les sept faisceaux vérité terrain ont pu être bien définis. Ces faisceaux contiennent les obstacles mentionnés plus tôt, soit des goulôts, des faisceaux se chevauchant, des régions ambiguës, et autre, dans le but de bien représenter les embûches qu'un algorithme de tractographie rencontre lors du tracking sur un cerveau humain.

Six acquisitions du fantôme ont été faites pour le concours, combinant des valeurs b de 2000, 4000 et 6000 s/mm^2 et des résolutions isotropiques de 3 et 6 millimètres. À noter que les fibres d'acryliques, plus larges que les fibres de la matière blanche, causent une FA ayant des valeurs autour de 0.1 à 0.2 [107], ce qui est bien en deçà des valeurs généralement trouvées dans la matière blanche [59]. De plus, le concours de 2009 demandait aux participants de ne soumettre qu'une seule tract par région d'intérêt, celle-ci étant comparée avec la vérité terrain. Cette méthode d'évaluation rend la comparaison entre les résultats du concours et des algorithmes de tractographie généraux un peu incertaine, car les algorithmes de tractographie

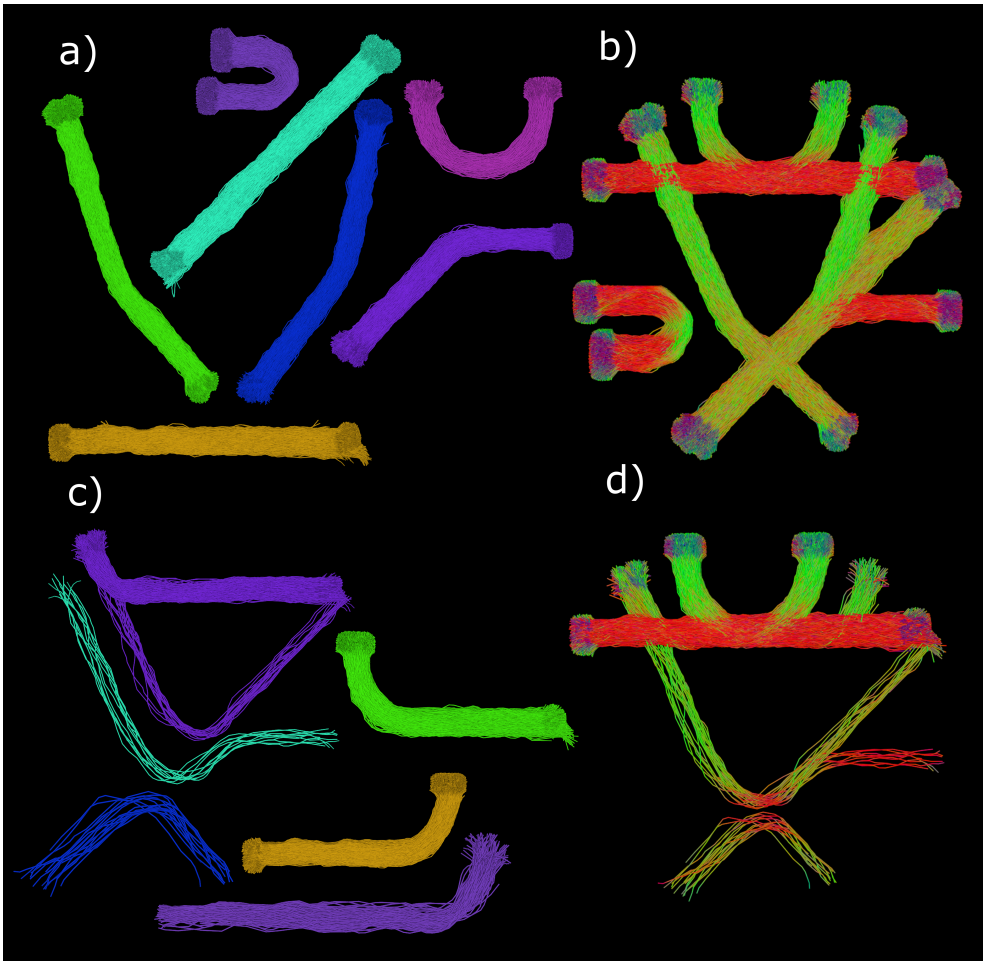

Figure 3.27 – Illustration de connections valides et invalides sur le *FiberCup*. a) Faisceaux valides (VB) connectant deux régions qui doivent être connectées. b) Ensemble des faisceaux valides formant les connections valides (VC). c) Faisceaux invalides connectant deux régions qui ne devraient pas être connectées (IB). d) Ensemble des faisceaux invalides formant les connections invalides (IC).

produisent généralement plusieurs tracts, non une seule.

Tractometer

Pour répondre aux insatisfactions face à l'évaluation originelle du FiberCup et mieux refléter la connectivité globale de la tractographie, le Tractometer [23, 24] a été développé et rendu publique. L'approche du Tractometer propose plutôt d'évaluer

les performances des algorithmes de tractographie, toujours sur le FiberCup, selon la connectivité de régions d'intérêts placées aux extrémités des faisceaux, afin de mimer l'interface GM/WM. Cette fois-ci, un masque de "WM" est aussi fourni, et l'initialisation des tracts est permise autant dans ce masque qu'à l'interface.

Quelques termes ont aussi été proposés afin de mieux exprimer la qualité d'un tractogramme reconstruit : le taux de *connections valides* (*valid connections*, VC) représente le pourcentage de tracts connectant les régions d'intérêt attendues. Le taux de *connections invalides* (*invalid connections*, IC) représente plutôt le pourcentage de tracts connectant des régions d'intérêt inattendues. Le taux de non-connections (*no-connections*, NC) représente quant à lui les tracts ne connectant pas deux régions d'intérêt à leurs extrémitées.

Des métriques sur les faisceaux reconstruits ont aussi été mises de l'avant, soit le nombre de faisceaux valides (*valid bundles*, VB) représentant le nombre de faisceaux attendus reconstruits. Inversement, le nombre de faisceaux invalides (*invalid bundles*, IB) représente le nombre de faisceaux reconstruits connectant deux régions qui ne devraient pas l'être. La figure 3.27 illustre quelques faisceaux valides et invalides reconstruits sur le *FiberCup*.

Le Tractometer propose aussi deux nouveaux ensembles d'acquisitions du FiberCup, soit avec des b-values de 650, 1500 et 2000 mm/s^2, chacune avec une résolution isotropique de 3mm. Les deux ensembles d'acquisitions ont aussi été moyennées, pour un total de neuf images de diffusion. Sur ces neuf images, plus de 57 000 tractogrammes ont été évalués et une analyse des résultats est disponible dans l'article originel [24].

ISMRM Tractography Challenge 2015

Le concours *ISMRM Tractograpy Challenge 2015* [93], organisé en conjonction avec la conférence ISMRM2015, eu pour but d'évaluer les algorithmes de tractographie sur des données synthétiques se rapprochant le plus possible de données réelles. Pour ce faire, un jeu de plusieurs tractogrammes fut généré à partir de données provenant du *Human Connectome Projet* [55], puis un radiologiste expert segmenta 25 faisceaux de ceux-ci. Ces faisceaux servirent à générer une image de diffusion et une image T1, toutes deux synthétiques, grâce à l'outil Fiberfox [101], auxquelles ont été ajoutées du bruit afin de rendre le jeu de donnée plus réel. La figure 3.28 illustre le processus.

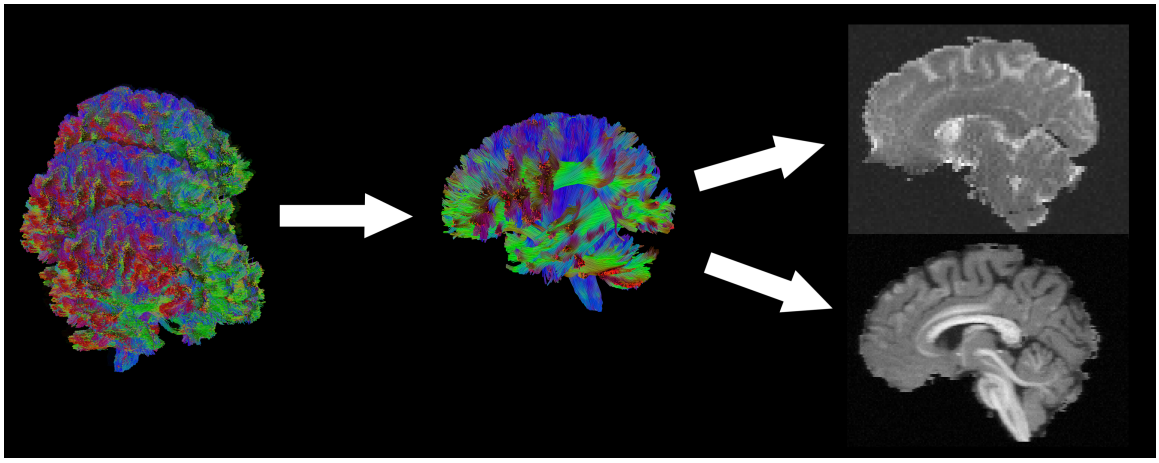

Figure 3.28 – Illustration de la création du jeu de données *ISMRM2015*. À gauche, plusieurs sujets du jeu de données *Human Connectome Project* [149, 147]. Au centre, les 25 faisceaux segmentés par un expert. En haut à droite, une image b0 tirée de la séquence de diffusion synthétique à une résolution de 2mm isotropique. En bas à droite, l'image T1 synthétique générée à une résolution de 1mm isotropique.

Afin d'évaluer les soumissions, le Tractometer a été utilisé et modifié afin de rapporter des métriques sur le chevauchement (*overlap*, OL) et le dépassement (*overreach*, OR) entre les faisceaux reconstruits et les faisceaux vérité terrain (c.f. fig. 3.29), en plus de celles rapportées par l'évaluation originelle. Un total de 96 tractogrammes ont été soumis, la plupart reproduisant 90% des VB et environ un tier de leur volume. Par contre, les tractogrammes soumis étaient majoritairement représentés par de faux positifs, illustrant les embûches rapportées plus tôt auxquels font face les algorithmes de tractographie.

3.2.7 Tractographie par apprentissage supervisé

La première section de ce chapitre a pu démontrer qu'une sélection incroyable de modèles est disponible à un utilisateur souhaitant représenter son signal de diffusion afin de le donner en entrée à un algorithme de tractographie. Certains modèles requièrent moins de données mais offrent une représentativité des croisements plus faible, et vice-versa. Nous avons aussi pu prendre connaissance de quelques problèmes affligeant les algorithmes de tractographie, ainsi que de quelques *a priori* anatomiques

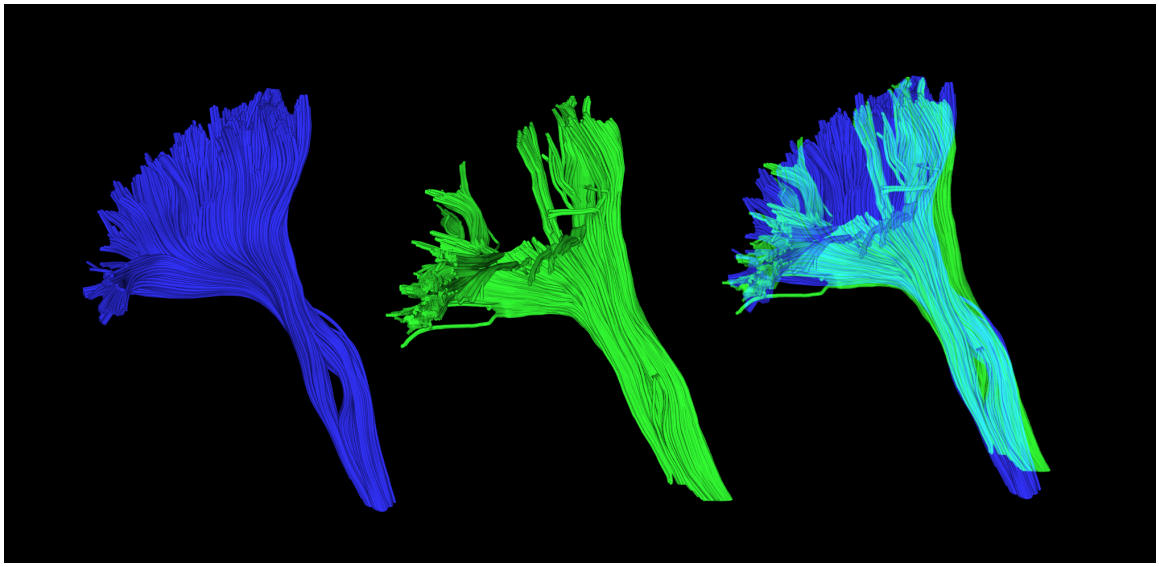

Figure 3.29 – Illustration du chevauchement entre un faisceau vérité-terrain et un faisceau reconstruit. Gauche) Vérité terrain. Centre) Reconstruction. Droite) Chevauchement (OL) entre les deux, en bleu pâle. Les parties en bleu foncées n'ont pas été reconstruites, alors que les parties en vert représentent le dépassement (OR) du volume du faisceau vérité-terrain.

et règles fait main possibles à ajouter aux algorithmes afin d'attaquer ces problèmes. Malheureusement, le choix de ces règles, heuristiques et cas spéciaux afin de gérer la propagation et l'arrêt de tracts rend l'implémentation et l'utilisation d'un algorithme de tractographie moderne bien complexe [46, 24, 93].

Tel que mentionné dans le chapitre 1, l'apprentissage supervisé réussi à extraire des motifs complexes dans les données d'entraînement afin de bien généraliser sur de nouvelles données en minimisant itérativement une fonction de perte dérivable. Dans le but d'aborder les problèmes auxquels la tractographie fait face, notamment le manque d'informations locales permettant de démêler les croisements, des algorithmes de tractographie par apprentissage supervisé ont vu le jour.

Bien des choix s'offrent à un concepteur souhaitant appliquer l'apprentissage supervisé à la tractographie : quelles seront les données en entrée ? Le signal de diffusion "brut" ? Ce même signal, mais ré-échantillonné selon un nombre de directions précis ? Les coefficients des harmoniques sphériques ? Le modèle d'apprentissage supervisé peut aussi avoir un grand impact sur le traitement de l'information de diffusion : est-ce

qu'un réseau de neurone servira à effectuer l'apprentissage ? Un modèle plus simple ? Ou plus compliqué ? Finalement, un éventail de possibilités est aussi offert pour la sortie du modèle : est-ce que le modèle prédira directement un vecteur d'orientation ou de direction ? Est-ce qu'une classification sera effectuée selon un nombre de directions prédéterminé ? Est-ce que l'algorithme arrêtera le tracking par lui-même ?

Dans cette sous-section, nous aborderons quelques configurations possibles des options présentées quant à l'implémentation d'un algorithme par apprentissage supervisé. Toutes les méthodes présentées ont en commun leur méthode d'évaluation : le *Tractometer*. À ce jour, il s'agit de l'outil le plus utilisé afin de quantifier les performances d'un algorithme de tractographie de par ses métriques représentatives ainsi que ses performances de référence et ses jeux de données publiquement disponibles . Pour un aperçu plus complet de l'état de l'apprentissage supervisé pour la tractographie, voir Poulin et al. [106].

Tractographie par Random Forest

Le premier effort afin de faire de la tractographie par apprentissage supervisé n'utilisera pas de réseaux de neurones, mais bien une forêt d'arbres décisionnels (*Random Forest Classifier*), où une multitude d'arbres décisionnels sont construits et entraînés sur des sous-ensembles de données légèrement différents pour ensuite prédire selon un vote majoritaire sur la sortie des arbres [100, 99].

À cette forêt est donné en entrée le signal de diffusion à la "tête" de la tract, soit la position courante du dernier point de celle-ci, ré-échantillonné à 100 directions et possiblement représenté par des coefficients d'harmoniques sphériques tel que présenté plus tôt. Dans le but de donner un peu plus d'information spatiale, le signal du voisinage est aussi donné en entrée en échantillonnant une demi-sphère (ou sphère si la tract n'a encore qu'un seul point) autour de la position et direction courante. Finalement, dans le but d'ajouter de l'information directionnelle, la dernière direction prise est aussi ajoutée au signal d'entrée.

La prédiction de cette forêt prend la forme d'une classification sur 100 directions également réparties sur une demi-sphère (ou sphère s'il s'agit de la première direction) face à la tract. Une 101e classe est ajoutée, correspondant à l'arrêt de la tract. La propagation de la tract peut se faire de façon déterministe, en retournant une somme

pondérée des classes par leurs probabilité donnée par le classifieur. Elle peut aussi se faire de façon probabiliste, en échantillonnant une distribution discrète formée par les classes et leurs probabilités associées. Finalement, la tract peut "rebondir" sur une paroi si un 2e vote pour la terminaison de la tract détermine qu'un point alternatif est préférable à l'arrêt réel de la tract.

Afin de valider la méthode, celle-ci a été comparée à 12 méthodes "classiques" sur le fantôme Fibercup et les soumissions originelles du concours ISMRM Tractography Challenge 2015. Sur ce dernier, deux expériences ont été tentées : un entraînement avec les tracts vérité-terrain, ainsi qu'un entraînement avec des tracts provenant d'une méthode classique déterministe utilisant un modèle par déconvolution sphérique sur le signal de diffusion brut (*CSD-DET*).

La méthode produit des résultats assez concluants sur le FiberCup : la forêt d'arbres décisionnels obtint des performances supérieures ou presque aux 12 autres méthodes classiques selon les 5 métriques évaluées. En comparaison avec les 96 soumissions du challenge ISMRM2015, la forêt d'arbres décisionnels entraînée sur les données d'un algorithme classique fut la seule à reconstruire 25 des 25 faisceaux possibles, tout en obtenant un chevauchement supérieur à la moyenne, en plus de dépasser largement les performances de l'algorithme ayant généré ses données d'entraînement. Tel qu'attendu, la méthode entraînée sur les données vérité-terrain performa encore mieux.

Bien qu'il est difficile de déterminer précisément comment la méthode réussit à obtenir des performances supérieures à celles qui ont généré ses données d'entraînement, les auteurs théorisent que l'information du voisinage et de directionalité, ainsi que l'accumulation du savoir provenant des données d'entraînement en sont en partie responsable.

Tractographie par réseaux récurrents

Alors que l'approche par forêt d'arbres décisionnels fut tout un accomplissement, certains *a priori* forts ont quand même été intégrés à l'algorithme. Notamment, l'arrêt implicite des tracts selon un vote, le rebond de la tract selon un autre vote ainsi que la dernière direction ajoutée au signal d'entrée. Dans le but d'avoir une approche purement axée sur les données, l'algorithme *Learn to Track* [104] a été proposé.

L'approche, basée sur l'apprentissage profond, propose d'entraîner un réseau de

neurones récurrent à prédire directement la prochaine direction plutôt qu'effectuer une classification sur celle-ci. Il est suggéré que les tracts ont des dépendances directionelles plus importantes que seulement sur la dernière direction. Donc, un réseau récurrent entraîné sur l'entièreté de la tract devrait arriver à mieux gérer les formes complexes des tracts, de mieux passer les croisements et démêler les goulots. En entrée, le réseau reçoit le signal de diffusion rééchantillonné sur 100 directions ainsi que, de par sa nature récurrente, sa "mémoire" interne. En sortie, le réseau effectue une régression sur la prochaine direction à prendre, plutôt qu'une classification sur celle-ci. Prédire directement la direction permet un contrôle plus fin de la tract et réduit les demandes computationnelles nécessaires par rapport à calculer et voter sur plusieurs directions. Contrairement à la méthode précédente, l'arrêt de la tract est géré par un masque de tracking plutôt que par l'algorithme pour simplifier l'apprentissage.

Tout comme la méthode précédente, le Tractometer a été utilisé pour évaluer les performances de *Learn to Track* sur les données du concours ISMRM2015. Dans le but d'avoir des résultats comparables, les tracts d'entraînement ont étés générées de la même façon, soit par un algorithme CSD-DET. De plus, afin d'offrir un référentiel, un réseau de neurones à propagation avant, non-récurrent, a aussi été entraîné de la même façon que le modèle récurrent. Un léger filtrage des tracts trop courtes et trop longues a été ajouté au pipeline afin d'améliorer les performances sans trop injecter d'a priori fort. Finalement, les deux types de réseaux de neurones ont aussi été entraînés en ayant aussi la dernière direction prise en entrée, afin d'injecter un *a priori* directionnel supplémentaire.

Malgré sa simplicité, le réseau de neurone à propagation avant réussit à atteindre des performances équivalentes ou légèrement supérieures aux résultats moyens des soumissions originelles du concours ISMRM2015. Par contre, le réseau de neurones ayant aussi la direction précédente en entrée atteint des performances bien pires que celui entraîné sans celle-ci. Les auteurs théorisent que le réseau réussit à atteindre ses performances qu'en copiant la dernière direction sans se préoccuper de l'information de diffusion, le causant à manquer beaucoup de connections.

Le réseau récurrent, par contre, réussit à bien se servir de l'information directionelle supplémentaire. Même sans celle-ci, le réseau récurrent atteint un ratio de connections valides supérieur à la moyenne ainsi qu'un nombre de faisceaux invalides bien

inférieur. En ajoutant la direction précédente, le réseau récurrent obtint un très bon chevauchement sans obtenir un dépassement dégénéré, contrairement aux soumissions originelles.

Dans des travaux subséquents, la procédure d'entraînement de *Learn to Track* fut adaptée pour n'apprendre qu'à reconstruire des faisceaux individuels plutôt que le cerveau au complet. Chaque réseau fut entraîné à reconstruire un seul faisceau à partir d'un jeu de données de 37 sujets ayant chacun cinq faisceaux manuellement segmentées. La méthode, nommée *Bundle-Wise Deep Tracker* [109], fut comparée à l'algorithme BST ainsi qu'à l'algorithme *Learn to Track* originel et obtint des performances supérieures aux deux méthodes selon toutes les métriques évaluées.

Tractographie par réseaux de neurones

Indépendemment de la méthode précédente, une autre approche, celle-ci entièrement par réseaux de neurones à propagation avant, a aussi été proposée afin d'obtenir un algorithme de tractographie indépendant de la modélisation des données [152]. Dans le but d'inclure de l'information sur le voisinage, l'entrée du modèle consiste en l'information de diffusion brute, encore une fois rééchantillonnée et représentée par les SH provenant de la déconvolution sphérique, mais cette fois-ci regroupée en un "bloc" $3 \times 3 \times 3$ autour de la tête de la tract. Le signal de neuf voxels sont donc donnés en entrée au modèle, auxquels sont ajoutés les quatre dernières directions de la tract afin d'ajouter de l'information directionelle au processus de tracking. Tout comme la méthode précédente, le réseau performe une régression sur la direction à prendre plutôt qu'une classification. Par contre, le tracking est cette fois-ci initialisé à l'interface WM/GM. Puisque le l'information de diffusion est supposée symétrique, le premier pas peut être ambiguë et sortir immédiatement du masque de tracking. Dans ce cas, le pas est inversé et le tracking se poursuit normalement.

Contrairement aux deux méthodes précédentes, les données d'entraînement ont été générées à partir de trois sujets provenant du *Human Connectome Projet* [55] avec l'algorithme *iFOD2* [141]. Les performances rapportées ont été calculées grâce au Tractometer : la méthode par réseaux de neurones permet d'obtenir un nombre réduit de faisceaux invalides et un très grand ratio de connections valides. Malheureusement, un chevauchement très restreint est aussi rapporté.

3.2.8 Conclusion

Dans ce chapitre, nous avons introduit le processus de *tractographie*. Nous avons commencé par décrire l'acquisition des données, en présentant l'IRM puis l'IRM de diffusion, puis comment l'information de diffusion peut être modélisée afin de bien représenter les structures sous-jacentes provoquant les contrastes observables. Nous avons ensuite décrit comment cette modélisation des données peut être utilisée afin de produire des *tractogrammes*. Nous avons vu un exemple simpliste d'algorithme de tractographie, quelques choix d'implémentations offerts ainsi que quelques améliorations possibles pour rehausser la plausibilité des tractogrammes reconstruits. Nous avons fait un survol léger des problèmes connus avec la tractographie, quelques méthodes permettant d'offrir des analyses comparatives des algorithmes disponibles pour terminer avec quelques tentatives de produire des algorithmes basés sur l'apprentissage machine.

Malgré les avancées proposées, quelques problèmes demeurent. Notamment, un manque criant de jeux de données force les concepteurs d'algorithmes de tractographie par apprentissage machine à devoir donner en entrée à leur modèle des tractogrammes provenant d'algorithmes classiques. Sans jeux de données exonérés des problèmes connus de la tractographie, les algorithmes de tractographie par apprentissage supervisé sont forcés à reproduire les problèmes décrits dans ce chapitre. Afin de palier à ces limitations, il est nécessaire de se tourner vers des méthodes indépendantes des données biaisées par les algorithmes classiques.

There is nothing wrong with not knowing, but there is something wrong with not learning.

—Dr. Sara Solla

Chapitre 4

Track-to-Learn : Un cadre général pour la tractographie par apprentissage par renforcement profond

Résumé

L'imagerie par résonance magnétique de diffusion est présentement la seule méthode non-invasive permettant d'évaluer la connectivité structurelle du cerveau. Depuis ses débuts, il a été vastement documenté que la tractographie est enclin à produire de fausses tracts tout en omettant de vrai-positifs. Des *a priori* anatomiques ont été conçus et implémentés à travers des algorithmes classiques afin de s'attaquer à ces problèmes, mais des problèmes demeurent et la conception et validation de ces *a priori* est très difficile. Récemment, des algorithmes par apprentissage supervisé ont été proposés afin d'apprendre des algorithmes de tractographie implicitement à partir de données brutes, sans dépendre d'*a priori* anatomiques. Par contre, ces méthodes sont dépendantes de données annotées très difficiles à

obtenir. Dans le but d'éliminer la dépendance à de telles données tout en tirant parti des capacités des réseaux de neurones, nous introduisons *Track-to-Learn* : Un cadre général pour la tractographie par apprentissage par renforcement profond. L'apprentissage par renforcement profond est un type d'apprentissage automatique qui ne dépend pas de données de type vérité-terrain, mais plutôt sur le concept de "récompense". Nous implémentons et entraînons des algorithmes afin qu'ils maximisent le retour d'une fonction de récompense basée sur l'alignement de tracts avec les directions principales extraites de données de diffusion. Nous montrons que des résultats compétitifs peuvent être obtenus sur des jeux de données connus et que les algorithmes sont capable de généraliser à des données inconnues bien mieux que les algorithmes par apprentissage automatiques précédemment introduits. À notre connaissance, il s'agit de la première utilisation fructueuse de l'apprentissage par renforcement profond pour la tractographie.

Contributions de l'article

— Nous proposons le cadre *Track-to-Learn* : un cadre général permettant de formuler la tractographie en tant que problème d'apprentissage par renforcement
— Nous analysons plusieurs composantes du cadre proposé afin de déterminer leur impact sur les tractogrammes reconstruits
— Nous montrons des résultats compétitifs lorsque comparé à des algorithmes de tractographie classiques et par apprentissage automatique
— Nous montrons des capacités de généralisation bien supérieures comparé aux algorithmes de tractographie par apprentissage automatique lorsque appliqué à de nouveaux jeux de données

Contributions des auteurs

— Développement du cadre et des méthodes implémentées et génération des résultats (Antoine Théberge)
— Idées et suggestions (Christian Desrosiers, Pierre-Marc Jodoin, Maxime Descoteaux, Antoine Théberge)
— Analyse des résultats et conception des figures (Antoine Théberge)
— Écriture de l'article (Antoine Théberge)
— Relecture et correction (Christian Desrosiers, Pierre-Marc Jodoin, Maxime Descoteaux, Antoine Théberge)
— Support et supervision du projet (Pierre-Marc Jodoin, Maxime Descoteaux)

Commentaires sur l'article

Cet article est le résultat de deux ans de travail acharné, où des hauts et des bas ont été rencontrés. Ce qui commença par un projet d'apparence simple frappa rapidement un mur. En effet, tel que mentionné dans l'article, nous utilisions dès le début des réseaux de neurones récurrents plutôt que leur contrepartie à propagation avant. Pour des raisons qui demanderont plus d'investigation, les agents entraînés n'arrivaient pas à démêler la "directionalité" d'une tract et étaient incapables de produire un tracking "arrière", malgré que le tracking avant soit bien reconstruit. Ce n'est qu'après un an de travail, l'implémentation de maint algorithmes d'apprentissage par renforcement, un changement de type de réseaux de neurones ainsi que bien des remises en questions que la méthode proposée fini par donner des résultats prometteurs, pour éventuellement égaler ou battre l'état de l'art. Finalement, l'article fut soumis pour publication à la revue *Medical Image Analysis* le 15 Novembre 2020, puis une révision fut présentée le 9 Février 2021.

Track-To-Learn: A general framework for tractography with deep reinforcement learning

Antoine Théberge
Département d'informatique, Université de Sherbrooke,
Sherbrooke, Québec, Canada J1K 2R1
antoine.theberge@usherbrooke.ca
Christian Desrosiers
Département de génie logiciel et des TI
École de technologie supérieure
Montréal, QC, CA, H3C 1K3
christian.desrosiers@etsmtl.ca
Maxime Descoteaux *
Faculté des Sciences
Université de Sherbrooke
Sherbrooke, QC, CA, J1K 2R1
maxime.descoteaux@usherbrooke.ca
Pierre-Marc Jodoin*
Faculté des Sciences
Université de Sherbrooke
Sherbrooke, QC, CA, J1K 2R1
pierre-marc.jodoin@usherbrooke.ca

Keywords: Tractography, Deep Learning, Reinforcement Learning

Abstract

Diffusion MRI tractography is currently the only non-invasive tool able to assess the white-matter structural connectivity of a brain. Since its inception, it has been widely documented that tractography is prone to producing erroneous tracks while missing true positive connections. Recently, supervised learning algorithms have been proposed to learn the tracking procedure implicitly from data, without relying on anatomical priors. However, these methods rely on labelled data that is very hard to obtain. To remove the need for such data but still leverage the expressiveness of neural networks, we

∗. Equal contributions. Co-senior authors

introduce Track-To-Learn: A general framework to pose tractography as a deep reinforcement learning problem. Deep reinforcement learning is a type of machine learning that does not depend on ground-truth data but rather on the concept of "reward". We implement and train algorithms to maximize returns from a reward function based on the alignment of streamlines with principal directions extracted from diffusion data. We show competitive results on known data and little loss of performance when generalizing to new, unseen data, compared to prior machine learning-based tractography algorithms. To the best of our knowledge, this is the first successful use of deep reinforcement learning for tractography.

4.1 Introduction

Tractography is the process of inferring white-matter structure from diffusion magnetic resonance imaging (dMRI), by using the diffusion signal to model the underlying fibre populations. From this information and initial seed points, the shape of white-matter tissue is iteratively reconstructed following local orientation information [18]. Three large families of tractography algorithms are available today, each computing the next tracking step differently. The deterministic algorithms always select the principal reconstructed direction. Probabilistic algorithms, by contrast, use the diffusion model as a probability density function for the next direction. Finally, global tractography algorithms try to iteratively infer all directions at once while minimizing a loss function [77].

However, deciding which tractography algorithm to use is not the only choice one has to make, as the method for modelling the diffusion signal also has an impact on the produced tractograms (i.e., the product of running a tractography algorithm) [71]. For example, diffusion tensor tractography [25] has been shown unable to properly represent crossing fiber populations [41]. Representing the diffusion signal using a more sophisticated model, such as the fiber orientation density function (fODF) [132], has been shown to provide fuller tractograms and overall more accurate fiber reconstructions.

Over the years, a plethora of tractography algorithms have been proposed in the hope of enhancing the coverage of the white-matter volume by streamlines without

generating too many false-positives (i.e., streamlines connecting regions that should not be connected). For example, some work has focused on providing the tractography algorithm with anatomically-informed termination critera [127], or on choosing the next direction by sampling a distribution of possible directions and selecting the most promising one [57], or on inferring streamline trajectories from an atlas of known pathways [160].

Some recent work has focused on adapting the diffusion model so it best fits the tractography process. Notably, Rheault et al. [116] proposed to retro-actively modify fODFs to redirect the tracking directions on a per-bundle basis. By formulating the diffusion signal as a noisy measurement, Malcolm et al. [97] proposed to use Unscented Kalman Filters [157] to estimate the parameters of a multi-tensor model according to the local signal and the streamline's previous directions, which can then be used to propagate the streamline further. In a similar fashion, [115] jointly estimate the parameters of an extended NODDI [163] model and use it to propagate streamlines using Unscented Information Filters [85].

Despite all these efforts, several issues remain as tractography is an *ill-posed* problem that tries to infer global connectivity only from local information [92]. Emerging from this observation is the "crossing-fibers" problem, where two bundles crossing at a low angle form a *narrow intersection* [114] and may confound the tracking algorithm into taking the wrong exit. Similarly, the "bottleneck" problem [114] emerges in areas where several white-matter bundles have to pass through the same "choke point" and then separate again. In these situations, tractography is typically unable to disentangle where to go after exiting the bottleneck. However, these problems do not purely come from directional decision making; they stem from the fact that, from a local point of view, every "exit" option makes as much sense as the others, resulting in a several false-positive bundles being generated and true-positives being omitted.

Recently, supervised machine learning (SL) methods have been proposed in the hope of learning the tracking procedure without the need of strong, hard-to-implement anatomical priors, as well as removing the need for diffusion modelling [106]. Harnessing the pattern matching abilities of machine learning, these methods propose to learn how to infer the next tracking direction directly from raw diffusion data and labelled data in the form of "ground-truth" tractograms. By learning directly from ground-truth data,

which is less likely to suffer from the previously-mentioned problem, these methods hope to better disentangle directions in problematic directions where local modelling alone is not enough.

While learning from labelled data seems appealing, generating these ground-truth tractograms is no easy feat. Even for hardened neuroimagery experts, inferring streamline orientation purely from a visual inspection of diffusion imaging is virtually impossible. One way to proceed is to build "phantoms", devices made of synthetic materials to mimic physical properties of the human brain, like the FiberCup [107, 108, 46]. Generating ground-truth bundles from these phantoms is doable as the synthetic bundle configuration is visible and tangible. However, being only ever so similar to real human brains, phantoms cannot fully confirm the capabilities of tractography algorithms. An alternative is to generate the most complete tractogram possible using a family of tractography algorithms and to try to reconstruct all true-positive connections with no regards for false-positives. Then, the overly-full tractogram can be virtually dissected and segmented by experts and/or bundle segmentation algorithms [93]. However, even when done by highly-trained experts following a strict protocol, a high inter-observer variability is observed [110, 136]. Because of the variability in segmentation and the algorithms used to generate them in the first place, this labelled data is heavily biased and is not necessarily free of the aforementioned problems. As such, very few datasets with ground-truth streamlines are publicly available today [119], preventing a standardized supervised learning process for tractography algorithms [106].

Therefore, while SL offers an appealing alternative to classical tractography algorithms, the very nature of diffusion MRI and tractography calls for methods that do not rely on ground-truth data.

Tangentially, deep reinforcement learning is a form of machine learning that does not require explicit labelled data, and has recently shown promising results in various fields, from robotic [129] to game playing [126, 145, 95]. Reinforcement learning formalizes the notion of trial and error by letting a learning agent experiment in its environment, providing feedback through a reward signal.

In this work, we propose to use deep reinforcement learning to leverage the power of deep learning for tractography without the need for hard-to-obtain ground-truth

fibers. Our main contributions are as follow:

1. We propose Track-to-Learn, a general framework to pose tractography as a reinforcement learning problem.
2. We perform an ablation study on several components of the proposed framework to shed light on their impact on the reconstructed tractograms and guide future works on tractography with reinforcement learning.
3. We demonstrate competitive results when compared to supervised machine learning and classical tractography algorithms.
4. We demonstrate competitive generalization capabilities to new, unseen-during-training datasets compared to prior supervised machine learning methods.

To our knowledge, we are the first to successfully use deep reinforcement learning for tractography. Code will be rendered public upon acceptance of this paper.

4.1.1 Related Work

Over the years, a few machine learning methods have been proposed to predict streamline directions directly from data. We review some of them in this section but for a more complete survey, we recommend the paper by Poulin et al. [106].

The first machine learning method used for tractography is the one by Neher et al. [100, 99] which employs a Random Forest classifier to decide on the next tracking step to make. The classifier performs a majority vote on 100 directions sampled from a half-sphere in front of the streamline, with a 101-th class added to model the termination of the streamline. As input, the classifier is given the raw diffusion signal at the head of the streamline, resampled to 100 directions, N signal samples in front of the streamline, and the last streamline directions followed by the model. As "ground-truth", streamlines generated from a constrained spherical deconvolution deterministic tracking algorithm (CSD-DET) [135, 132] on the ISMRM2015 Tractography challenge dataset [93] were used, as well as the ground-truth streamlines released as part of the dataset. Results were computed on the same dataset, using the Tractometer [23, 24] evaluation tool. By comparison to the original ISMRM2015 submissions, the Random Forest was the only one able to reconstruct all 25 bundles from the ground truth as well as to achieve a higher-than-average overlap.

In order to learn streamline tracking more directly from the data, the Learn-to-Track algorithm [104] was proposed. The authors argue that streamlines have directional dependencies which go beyond the last streamline segment. To this extent, they use recurrent neural networks (RNN), more specifically Gated-Recurrent Unit [28] networks *. By using RNNs, the authors condition the next tracking step not only on the local signal, but also on the streamline's previous directions, which helps the bottleneck and crossing-fibers problems. As input to their network, the authors also used the raw diffusion signal resampled to 100 directions. Contrary to Neher et al. [100, 99], the RNN outputs the new tracking step directly, without having to perform a classification on fixed directions. The method was also trained using tractograms generated by a CSD-DET algorithm, and managed to outperform most of the original ISMRM2015 tractography challenge submissions.

Similarly to Learn-to-Track, Benou et al. introduced the DeepTract algorithm [19]. The algorithm also uses RNNs but instead outputs a conditional (on the RNN's internal representation) orientation distribution function (ODF) learned from reference tracks. DeepTract then performs classification on the conditional ODFs, much like Neher et al. [100, 99], which allows them to perform deterministic as well as probabilistic tractography. They report competitive results when compared to the ISMRM2015 Tractography Challenge submissions and other ML-based tractography methods when trained using either ground-truth data or tracks generated from MITK [158].

Analogously, Wegmayr et al. proposed the iFOD3 [152] method which uses a multilayer perceptron instead of a RNN. As with the other methods, the authors use the raw resampled signal. However, to add information about the neighborhood of the head of the streamline, a bloc of 3×3×3 voxels is fed to the network instead of the signal coming from a single voxel. Also, because multilayer perceptrons have no temporal memory, the authors added the last four directions as input to their neural network. While the other methods relied on seeding from the white-matter, Wegmayr et al. chose to seed at the white-matter/grey-matter interface. Furthermore, contrary to other methods, the authors chose to generate streamlines from the iFOD2

∗. By comparison to their feed-forward alternative, RNNs produce an internal representation of the input data alongside each prediction. This internal representation is then fed to the next input so that it can be reused, allowing the network to build a memory of past inputs and predictions.

algorithm [133]. Results from the Tractometer evaluation tool revealed a high valid connection rate and a low number of invalid bundles, but also a very low overlap.

Subsequently, Wegmayr et al. proposed Entrack [151]: a probabilistic machine-learning model for tractography. Building on their previous method, the output of the model is a Fischer-von-Mises (FvM) distribution instead of a deterministic output. The algorithm allows for probabilistic tracking and uncertainty quantification. This results in increased robustness, reduced overfitting and generally better performance compared to their previous work. However, the method still is dependent on labelled-data and does not outperform its reference classical method.

Similar to our approach, an RL method using expanding graphs was first proposed to reduce false positives [155]. The method used ground-truth bundles from the ISMRM2015 Tractography Challenge dataset to infer starting points and end goals for the tracking. It works by generating an expanding graph to discover paths linking the starting and goal regions, and learning a value for each node of the graph using Temporal Difference Learning [118]. Then, the optimal path connecting the starting and ending regions can be inferred from the learned value function. Unfortunately, the reward function remains undisclosed. While this method is interesting, it presumes the existence of ground-truth data, or at least starting and ending regions, which are not available in clinical settings. The method also uses positions as input states, which makes the algorithm non-robust to rotation or translation in the diffusion data. Finally, it is unclear if the algorithm is trained on a per-bundle basis or learns a value function for the whole brain.

4.2 Method

4.2.1 Preliminaries

Reinforcement learning is a framework that allows to solve problems via a sophisticated trial and error process, by placing a learning agent in an environment and providing it with a reward signal based on actions performed. The agent then tries to adapt its strategy to maximize its expected cumulative reward. This trial and error process is formalized as an infinite-horizon Markov Decision Process (MDP) defined

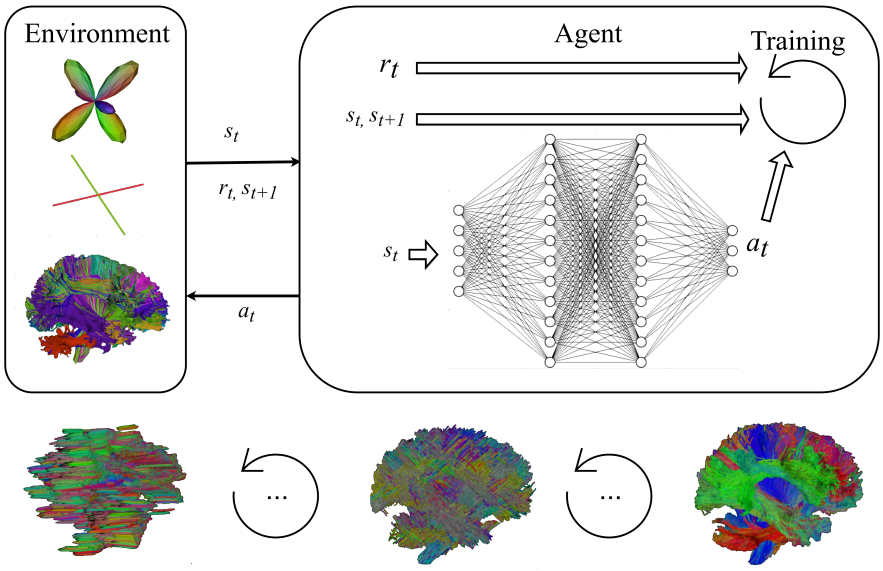

Figure 4.1 – The reinforcement-learning-for-tractography loop as well as the evolution of produced tractograms. Top part: To the left, the environment produces a state s_t from the diffusion signal (see section 4.2.2 for a description) as well as a reward r_t, which is computed from the last tracking step and the peaks extracted from the fODFs at the streamline's head. Both are given to the agent: the state is used as input to the policy (a neural network) to produce an action a_t, in this case a new tracking step, and the reward is used to train the agent to produce better actions. The environment receives the action a_t, normalizes it to the chosen step size and computes the new position of the streamline's head. A new state s_{t+1} and new reward r_{t+1} is returned and the cycle continues. Bottom part: Evolution of the tractograms produced by agents during training.

by the tuple (S, A, p, r), where S is the space of all possible states and A is the space of all possible actions. p indicates the transition probability $p(s_{t+1}|s_t, a_t), s_t, s_{t+1} \in S$, $a_t \in A$, the probability of landing in state s_{t+1} by executing action a_t in state s_t at time t. $r(s_t, a_t)$, usually shortened to just r_t, is the reward obtained from the environment by executing action a_t in state s_t at time t. The agent will have a policy $\pi(a|s)$ which denotes the probability of taking action a at state s.

RL algorithms typically use the following loop: An initial state s_0 is generated by the environment. An action a_0 is then generated according to the agent's policy $\pi(a_0|s_0)$ and sent back to the environment so that a new state s_1 is computed together

with a reward r_0. Afterwards, the agent uses the next state s_1 to produce action a_1, then a new state s_2 and a new reward r_1 are returned by the environment, and so on. The loop continues for a number T of timesteps, or until a stopping criterion is met according to the environment. The series of states and actions that occur is called an *episode*. At the end of an episode, the environment is reset to an initial state and a new episode starts.

The overarching objective of RL is to find the optimal policy $\pi_\theta(a|s)$ that maximizes the expected sum of future rewards. The RL objective is often formulated as:

$$J(\theta) \;=\; \sum_{t=0}^{T} \mathbb{E}_{r\sim\pi_\theta}\Big[r_t\Big], \tag{4.1}$$

where θ are the parameters of the policy π. In that perspective, it is common to define the *value function*, i.e. the expected reward in state s_t and time t:

$$V_{\pi_\theta}(s_t) \;=\; \sum_{k=0}^{T-t} \mathbb{E}_{\pi_\theta}\Big[\gamma^k r_{t+k}\Big], \tag{4.2}$$

where $0 \ll \gamma < 1$ is a discount factor preventing the cumulative reward to move towards infinity in the case of episodes with infinite length. The discount factor has the property of promoting short-term reward when adjusted towards lower values and long-term reward when adjusted towards higher values. Another commonly-used function is the *state-action value function*, or simply Q-function

$$Q_\pi(s_t, a_t) \;=\; r(s_t, a_t) + V_\pi(s_{t+1}), \tag{4.3}$$

which is the expected upcoming reward that one would get by selecting action a_t at state s_t and then following the policy π in subsequent states. To determine which action is best in which state, RL algorithms tend to try to maximize the Bellman Equation defined as

$$V_\pi(s_t) \;=\; r_t + \max_{a_{t+1}} \; \gamma Q_\pi(s_{t+1}, a_{t+1}). \tag{4.4}$$

Assuming we have the optimal Q-function, that is the state-action value function that

produces the true expected return for the optimal policy, then the optimal policy only consists of iteratively selecting the action that maximizes the Q-function. Finding this optimal pair of evaluation function and policy is the goal of reinforcement learning.

4.2.2 Proposed framework

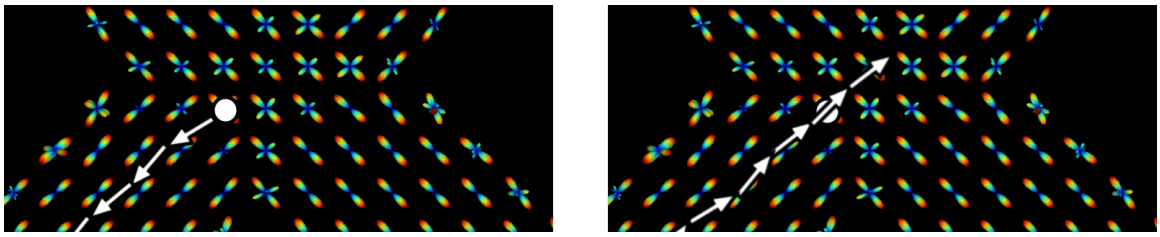

Figure 4.2 – "Forward" and "backward" tracking processes. *Left*: the "forward" environment initiates a seed (white dot) and the tracking starts in one direction, indicated by arrows. Once the tracking is over, the half-streamline is flipped and sent to the backwards environment. *Right*: the half-streamline is flipped, the initial seed point (white dot) becomes the head of the streamline, and the tracking continues.

In this work, we formulate the tractography process as a reinforcement learning problem. The role of the agent is to produce actions (i.e., tracking steps) according to the state of the environment, which is based on the diffusion signal. The role of the environment is to keep track of streamlines, produce states and receive tracking steps to propagate the streamlines. The environment is also in charge of providing the agent with a reward signal, computed according to the tracking steps and the states encountered, and to handle tracking termination.

The RL loop goes as follows: At first, a tracking seed is generated. The diffusion signal s_0 at the seed position is sent to the agent. The agent predicts the tracking step a_0, which is sent back to the environment to be used to propagate the streamline further. The environment calculates the new position of the streamline's head. The tracking direction is compared against the local peaks and a reward r_0 is sent back to the agent, along with state s_1. The loop continues until the tracking goes out of the tracking mask, or the angle between two consecutive segments is too high. Once an episode is over, the environment returns to the user the resulting streamline.

The framework allows users to provide their own signal volume (c.f. section 4.2.2), peaks volume (c.f section 4.2.2), seeding mask, tracking mask and tracking step-size so they can build their own environment. Figure 4.1 illustrates the proposed framework. The following sections describe each component in details and how the framework is implemented.

Environment

In RL, the role of the environment is to produce initial states, compute new states from actions, handle episode termination, and produce rewards. As such, in our context, the environment produces initial states from seeds generated from a seeding mask. The states correspond to the signal (described in section 4.2.2) at the current streamline's *head*, or last position. The state is sent to the agent, which produces an action $\mathbf{a}$, i.e., a tridimensional vector which is the next tracking direction. The environment then re-scales the action to the tracking step-size Δ such that

$$\widehat{a} = \frac{\mathbf{a}}{||\mathbf{a}||}\Delta \tag{4.5}$$

where $\widehat{a}$ is the rescaled tracking step, and computes the next streamline's position, which leads to computing the next state. The reward of the current timestep is also computed, as well as a marker d_t indicating whether or not the tracking for the current streamline is over (cf. section 4.2.2). Streamline termination depends on the following conditions:

1. The streamline exits the white-matter mask;
2. The streamline exceeds a predetermined maximum length;
3. The angle between two consecutive segments in the streamline exceed a maximum angle.
4. The streamline exceeds a maximum cumulative angle between segments.

The angle between streamline segments is defined as

$$cos^{-1}\langle \mathbf{u}_i, \mathbf{u}_{i-1} \rangle \tag{4.6}$$

and $\mathbf{u}_i$ are normalized streamline segments. If none of these conditions is satisfied, tracking continues and the tuple (s_{t+1}, r_t, d_t) is returned to the agent.

In itself, tractography is a two-step process. Because diffusion is by nature symmetric, tracking has to first follow one "direction" of the diffusion signal, and then go back to its original position to follow the "other direction". As such, and distancing ourselves from the standard RL setting, two environments are needed. The first one (or *forward*) handles streamlines from initial seeds and produces "half-streamlines". The second (or *reverse*) environment takes in half-streamlines and reverses them so that the initial seed point becomes the head of the streamline. Note that "forward" and "reverse" here do not imply a specific direction, such as anterior or posterior, but only that they follow the diffusion signal in different directions. To allow a form of self-supervised learning by the agent, tracking is still performed on the reversed streamlines starting from the new initial points, states are still sent to the agent, and rewards are calculated according to the new actions †. However, to preserve the integrity of the flipped half-streamlines and to guide the tracking process, actions that are computed by the agent while the agent is retracing its steps are discarded and replaced with the pre-existing half-streamlines segments. When the reversed half-streamline has been reconstructed, tracking continues as normal. The process is illustrated in figure 4.2.

To take advantage of the batching abilities of neural networks as well as to simulate multiple agents learning at the same time, multiple streamlines are initialized by the "forward" environment and handled at the same time. As such, a list of initial states $s_0^0, s_0^1, ..., s_0^N$ where N corresponds to the batch size, is sent to the agent. The agent then answers with a batch of $a_0^0, a_0^1, ..., a_0^N$ actions which results into a series of N new states and N rewards and the cycle goes on. As individual streamlines terminate, their corresponding states are removed from the batch and are appended to a tractogram. When all the forward half-streamlines are terminated, the streamlines in the tractogram are flipped and fed to the backward environment.

To align ourselves with most of the recent work done in reinforcement learning, the

†. We empirically found that the "re-tracking" process helped the agent learn to perform tractography in both "directions". However, further investigation is needed to properly measure the impact of the procedure.

environment is implemented following the OpenAI Gym [9] specification. The forward environment exposes the *reset* method, which takes in no parameter and returns the initial state s_0, as well as the *step* method, which takes in the action a_t and returns the next state s_{t+1}, reward r_t and completion signal d_t. The backward environment follows the same specification, but its *reset* method takes in half-streamlines that have been flipped.

Input signal

The strict Markovian requirements of off-the-shelf RL models do not apply to brain tractography. First, RL algorithms typically assume a Markovian process of order 1, where any state s_t depends only on the state and action s_{t-1}, a_{t-1} before it. This principle is named the Markov Property. This property is not respected when the transition probabilities correspond to $p(s_t|s_{t-1}, a_{t-1}, ..., s_k, a_k)$ with $k < t-1$. As illustrated by figure 4.2 and mentioned by Poulin et al. [104], streamlines have directional priors that go much beyond just local information. As such, the action a_t for propagating the streamline in a certain direction cannot be only driven by the local information contained in s_t, but by the last couple of states/actions the system has visited/made before.

Second, RL assumes that state s_t contains all of the information needed to perform the right action a_t. Unfortunately, the true state of a brain structure amounts to the sub-pixel cellular structure of the white-matter which is unavailable non-invasively. *In-vivo* tractography relies on diffusion MRI data which amounts to an observation o_t of the true state s_t. This is much like playing a video game by looking at the screen: the observations fed to the agents are the pixels displayed while the true state lies in the console registers.

All in all, our proposed model is a Partially-Observable Markov Decision Processes (POMDP) of order greater than 1. POMDPs are defined by the tuple (S, A, p, r, O, Ω), where S is the set of all true states the environment can have, A the set of all actions possible, p the transition probabilities as described before, r the domain of possible rewards, O the set of all possible observations and Ω the observation probabilities $\Omega(o_t|s_t, a_t)$.

While the framework allows for any diffusion signal representation to be used

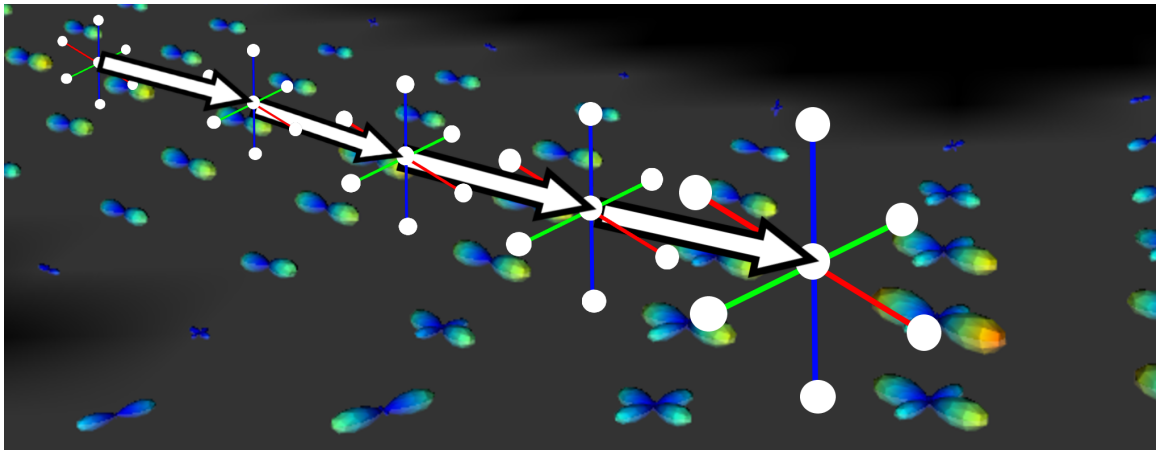

Figure 4.3 – Illustration of the input signal as well as the neighbouring, directional and mask information. SH coefficients are represented by fODF glyphs, the tracking mask is grey where tracking is allowed and black where it is not. Streamline segments are represented by arrows. White circles represent where the input signal is computed.

as observations fed to the agent, such as the raw resampled diffusion signal, ball-and-stick [7, 73] or even diffusion tensors [18], we adopted a different approach. As illustrated in figure 4.3, the input signal fed to our agent are spherical harmonic (SH) coefficients representing fiber ODFs [132, 33] obtained by a constrained spherical deconvolution [132]. To add spatial information, the signal from the six immediate neighbours (up, down, left, right, front, back) at the streamline's end is appended to the input. To add further contextual information, the tracking mask values at these neighbouring points are also appended to the signal. Finally, to alleviate the effects of the directional dependencies on streamlines and inject additional directional information, the last four streamline segments are appended to the input signal, much like Wegmayr et al. [152]. Although the framework can use an arbitrary number of previous directions to be used as part of the state, we empirically found that 4 worked best for our needs. All things considered, we end up with a 215-dimensional input: 28×7 SH coefficients + 7 mask values + 4×3 vector components for the previous directions.

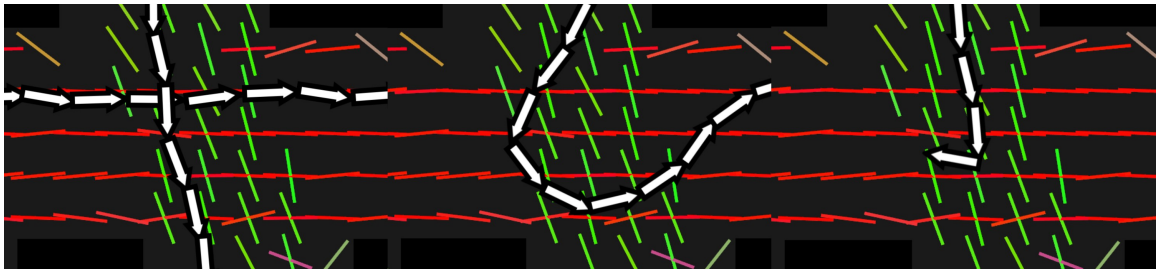

Figure 4.4 – Illustration of the reward signal. The green and red sticks are ODF peaks and the white arrows are streamline segments. *a*) Multiple streamline segments are well aligned with peaks extracted from the fODFs and therefore receive high reward ($0 \ll r_t < 1$); *b*) Multiple streamlines segments are badly aligned with the peaks and do not receive a high reward ($0 < r_t \ll 1$); *c*) Streamline segments have good alignment with the peaks, however, the last two segments have a high angle between them, bringing the reward at that point closer to -1 and terminating the tracking process.

Reward function

To guide the learning process, a suitable reward function must be provided to the agent. Because the agent is not given any explicit goal, the reward function (and its maximization by the agent) must encompass what the true underlying goal is, or what the desired behaviour of the agent should be. Defining an explicit goal to tractography is unclear. As mentioned before, the exact reconstruction of the ground-truth data should not be an end by itself, as we do not have access to it.

For our implementation of the proposed Track-To-Learn framework, we take a look at the behaviour of classical deterministic tractography algorithms for inspiration: how do they perform tractography? From their local model, they extract maxima and choose the one most aligned with their previous direction, propagate the streamline forward in the chosen direction and repeat the process until the angle with their previous direction is too high or tracking goes out of the tracking mask. Therefore, if we want to learn a tractography algorithm, we should promote behaviours that are executed by one.

We define the reward function for our environment as follows: the reward function is the absolute[‡] cosine distance between the action performed by the agent and the

‡. Diffusion MRI signal being symmetric, peaks can be followed in any direction.

peak extracted from fODFs most aligned with it, multiplied by the cosine distance between the last streamline segment and the action performed:

$$r_t = |\max_{\mathbf{p}_i} \langle \mathbf{p}_i, \mathbf{a}_t \rangle| \cdot \langle \mathbf{a}_t, \mathbf{u}_{-1} \rangle \tag{4.7}$$

where $\mathbf{p}_i$ are the peaks at the head of the streamlines, $\mathbf{a}_t$ is the action produced by the agent at time t, $\mathbf{u}_{-1}$ is the last streamline segment before the action is appended to it and $\mathbf{p}_i, \mathbf{a}_t, \mathbf{u}_{-1}$ are all normalized. This allows the agent to produce streamlines that are aligned with the underlying signal while enforcing streamline smoothness. An illustration of the relation between the reward function and streamlines is available in figure 4.4. While we chose to use peaks extracted from the fODFs used as input signal (cf. section 4.2.2), the framework allows for any directional information in the form of local maxima to be used, such as peaks extracted from diffusion ODFs [143] or even eigenvectors from diffusion tensors.

Agents

To train agents to produce the next tracking step, we chose two algorithms: Twin-Delayed Deep-Deterministic Policy Gradient (TD3) [47] and and Soft Actor-Critic (SAC) [66].

TD3 We selected TD3 for its simplicity and high performance on robotic control tasks. The TD3 algorithm uses the Actor-Critic framework, where a policy network (the actor) performs actions and a critic network infers what the expected value of the actions performed by the actor will be. The "Twin" part in TD3 comes from the fact that it uses *two* critics instead of one to fight overestimation of the state-action values due to noise in the data [58]. The actor predicts deterministic actions, a 3D vector indicating the next unnormalized tracking step. To allow for exploration, TD3 requires noise to be added to actions at training time such that

$$\mathbf{a}'_t = \mathbf{a}_t + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_{train}\mathbf{I}) \tag{4.8}$$

where $\mathbf{a}_t$ is the action at timestep t and σ_{train} is the *exploration rate*, a hyperparameter controlling the exploration noise. In parallel, the critic outputs a single scalar representing the expected value of the action performed by the actor.

SAC Analogously, we chose the SAC algorithm for its similarity to TD3 with the added benefit of it having a stochastic policy. This implies that the actor predicts two 3D vectors respectively representing the mean and variance matrix of a distribution, instead of actions directly. SAC then samples actions from the Gaussian distribution formed by the two outputs. This allows us to remove the need for explicitly controlling the exploration done by the agent during training (more on this in section 4.5.6). SAC also employs the Actor-Critic framework and also uses two critics instead of one. Finally, to prevent the policy from "defaulting" to a specific action given an input state, SAC reformulates the RL objective previously presented in equation 4.1 as

$$J(\theta) = \sum_{t=0}^{T} \mathbb{E}_{r\sim\pi_\theta}\Big[r_t + \alpha\mathcal{H}[\pi_\theta]\Big], \tag{4.9}$$

where $\mathcal{H}[\pi_\theta]$ is the entropy of the policy and α is a hyperparameter controlling the input of the entropy to the new objective. Besides the output of the actor, the architecture employed by SAC is the same as TD3.

We also chose these two algorithms for their deterministic (TD3) and stochastic (SAC) nature which echoes conventional deterministic and probabilistic tracking methods.

Tracking noise To promote a better coverage of the white-matter at test time, noise proportional to the FA is added to the actions such that

$$\mathbf{a}'_t = \mathbf{a}_t + (1 - \mathrm{FA})\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_{test}\mathbf{I}) \tag{4.10}$$

where $\mathbf{a}_t$ is the action at timestep t, FA is the fractional anisotropy value at the head of the streamline and σ_{test} is the tracking noise. We chose this strategy instead of applying the same amount of noise everywhere because we presume the algorithm to benefit more from exploratory noise in the more ambiguous, lower FA regions (such as

crossings) while lowering the risk of ending the tracking prematurely in simpler, high FA regions because of noise. Empirically, we found that this process improves bundle coverage without causing streamlines to end prematurely too much, whereas adding uniform noise (with no regards for the FA) causes streamlines to exit the tracking mask when they shouldn't much more. Even though SAC employs a stochastic policy and therefore could provide its own noise at test time, we empirically found that the trained agents performs better when their policy is used deterministically at test time and tracking noise is explicitly added as presented above (cf. section 4.3.1 for more details).

Implementation details Models for both algorithms are three-layer fully-connected neural networks with a width of 1024 neurons. Both algorithms were implemented using the PyTorch [105] framework. Hyperparameters were chosen on a per-experiment and per-agent basis through a Bayesian hyperparameter search using comet.ml [§], by arbitrarily selecting the agents with the best performance according to the evaluation metrics defined in section 4.3.2. Selected hyperparameters for each experiments are available in 4.A.

4.3 Experimental protocol

4.3.1 Benchmarking & Experiments

Here we present experiments designed to assess the reconstruction and generalization capabilities of our method.

Experiment 1: Performance on the FiberCup dataset

To assess the performance of our agents, we trained it and then track on the FiberCup dataset. The FiberCup [107, 108, 46] is a synthetic dataset that replicates a coronal slice of the brain. The dataset contains 3 axial slices acquired on 30 directions with a b-value of 1000 and an isometric voxel size 3mm. The 7 groundtruth bundles present challenging configurations such as fanning, crossing and kissing fibers.

§. `https://comet.ml`

No further preprocessing of the data was necessary as the simulated version of the FiberCup is already free of artifacts. From the data, a volume of SH coefficients, a fractional anisotropy (FA) map and peaks from the fODFs were produced using dipy [50] and scilpy [¶], and a tracking and seeding mask was extracted by thresholding the FA map.

To assess the generalization capabilities of our method, we also track (without re-training) on a copy of the FiberCup dataset that has been flipped horizontally, aptly dubbed "Flipped". This allows us to measure to which extent our agents can generalize in an environment where the signal distribution is essentially the same, but the configuration of the bundles is different.

To provide a baseline comparison, we trained and tracked on the same datasets with the Learn-to-Track [104] algorithm, using ground truth bundles as reference. Two versions of Learn-to-Track were trained: one with raw diffusion signal as input, like the original paper, and the other with fODF SH coefficients and WM mask as input, like our method. The implementation and hyperparameters were provided by the original authors.

To provide an even more thorough comparison, we compare our trained agents to two classical algorithms [‖]: CSD-DET and CSD-PROB. Both algorithms use fODF SH coefficients as input and respectively either extract peaks and follow them deterministically or use the fODF as probability density function on the next direction to take.

Training was performed at 2 seeds per voxel on the whole white-matter mask, and testing was performed at 33 seeds per voxel. Tracking was performed with a step-size of 0.75mm, a minimum streamline length of 20mm and a maximum streamline length of 200mm. Hyperparameters for this experiment are available in appendix 4.A.1.

Experiment 2: Performance on the ISMRM2015 dataset

This experiment was designed to mirror state-of-the-art methods by Neher et al. [100, 99], Poulin et al. [104] and Benou et al. [19], by training and testing Track-To-Learn agents on the ISMRM2015 dataset. The ISMRM2015 Challenge dataset [93]

¶. `https://github.com/scilus/scilpy`

‖. Implementation available at `https://github.com/scilus/scilpy`

is a more realistic synthetic dataset. It was generated by tracking on multiple subjects from the Human-Connectome Project dataset [55], and the set of tractograms was manually cleaned and segmented into 25 bundles representing both the major pathways of the white-matter as well as smaller, harder to track bundles. The Fiberfox [101] tool was then used to generate a diffusion-weighted volume as well as a T1 image from the 25 bundles. As such, the dataset does have a proper ground-truth but is still considered synthetic. We chose to preprocess the dataset using Tractoflow [139, 79, 42, 50, 142, 6, 68], which lead to an SH volume of 1×1×1mm resolution. The FA and peaks were also computed by Tractoflow, and the tracking and seeding mask was also obtained by thresholding the FA map and then manually cleaning the mask. However, contrary to prior methods, no ground-truth streamlines or masks were used in the process of training.

As they all presented a similar experiment, we provide comparison against Neher et al. [100, 99], Poulin et al. [104], Benou et al. [19] and Wegmayr et al. [151], by extracting available metrics from their respective papers. We also report the mean results of the original ISMRM2015 White-Matter Tractography challenge [93] submissions. Finally, we also provide a comparison against the same classical methods presented in section 4.3.1.

Training was performed at 2 seeds per voxel on the whole white-matter mask, and testing was performed at 7 seeds per voxel. Tracking was performed with a step-size of 0.75mm, a minimum streamline length of 20mm and a maximum streamline length of 200mm. Hyperparameters for this experiment are available in appendix 4.A.2.

Experiment 3: Generalization through HCP and ISMRM2015 datasets

We again explore the generalization capabilities of the proposed method, however this time on a more complex dataset. As with prior work, we train our agents on the HCP dataset and test it on the ISMRM2015 dataset. The HCP 1200 dataset [149, 148, 147] is a staggering collection of 1200 healthy subjects acquired on a 3T MRI scanner. The diffusion data was acquired at a 1.25mm isometric voxel size for 270 directions distributed over 3 shells. For our project, we used a single subject (ID: 100206) from the HCP 1200 test-retest subset, which was processed with Tractoflow to obtain similar inputs as with the ISMRM2015 dataset. As opposed to the other

datasets, the HCP dataset does not have a publicly available set of "ground-truth" streamlines. However, because our method does not rely on labelled data, we could still train on it. Like the ISMRM2015 dataset, training was performed at 2 seeds per voxel and testing was performed at 7 seeds per voxel.

To provide a baseline comparison, we train Learn-to-Track [104], as in Experiment 1. However, because the HCP dataset is *in-vivo* and therefore cannot provide ground-truth streamlines, we train it on the HCP Young Adult dataset [56] using reference tracks from TractSeg [156]. Subjects $749361, 816653, 814649, 871762, 987983$ were used as training set for Learn-to-Track and subjects $704238, 896879$ for validation. As they also presented a similar experiment, we provide comparison against Neher et al. [100, 99], Wegmayr et al. (2018) [152] and Wegmayr et al. (2020) [151], by extracting available metrics from their respective papers.

Training was performed at 2 seeds per voxel on the whole white-matter mask, and testing was performed at 7 seeds per voxel. Tracking was performed with a step-size of 0.75mm, a minimum streamline length of 20mm and a maximum streamline length of 200mm. Hyperparameters for this experiment are available in appendix 4.A.3.

In further experiments, we provide analysis for some of the proposed method's hyper-parameters and components.

Experiment 4: Impact of the discount parameter

In this experiment, we explore the impact the discount parameter (γ) has on the Tractometer metrics. As mentioned in section 4.2.1, γ prevents the sum of rewards to go towards infinity. From a RL point of view, it allows to control how "greedy" the agent can be, or how much importance it gives to rewards that might be produced only far in the future. From a tractographic point of view, we hypothesize it might allow to control how much the agent is willing to go through "dangerous" regions in the white-matter to continue tracking or, on the opposite end, it might make the agent "give up" early.

To analyze the impact of this parameter, we picked our TD3 agent from Experiment 1 and performed multiple training runs on the FiberCup dataset while having all hyperparameters fixed except γ.

Experiment 5: Impact of the exploration noise

In this experiment, we analyze the impact of the exploration noise on the Tractometer metrics. Exploration is an important concept in RL: through the process of trial and error, a learning agent has to explore its environment by sometimes trying actions that may not be better than what the agent currently estimates as optimal. If an agent does not explore enough, it might never find actions that are better than what it currently knows. However, if an agent explores too much, it might not receive as much reward as it should and the learning could stall. This is aptly named the *Exploration vs. Exploitation* problem in RL.

While SAC handles exploration on its own through its stochastic policy, TD3 requires explicit noise $\epsilon \sim \mathcal{N}(0, \sigma_{train})$ to be added to actions performed by the agent during training. As such, the σ_{train} parameter must be carefully chosen as to allow the agent to discover which actions should be performed at different places in the white-matter, without adding too much noise so that the agent cannot perform optimal actions and reinforce them.

To analyze the impact of this hyperparameter, we performed multiple training runs on the FiberCup dataset with the TD3 algorithm and fixed all hyperparameters except σ_{train}.

Experiment 6: Impact of noise at test time

As mentioned in section 4.2.2, noise proportional to the FA is added at test time to promote white-matter coverage (c.f. equation 4.10). By doing this, we can control how the tracking behaves without having to retrain an agent. We use the Track-To-Learn SAC agent trained for Experiment 2 and perform tracking, without re-training, on the ISMRM2015 dataset with various levels of noise σ_{test}. Because SAC employs a stochastic policy, we can also track using its own probabilistic output instead of artificially adding noise.

Experiment 7: Impact of reward on performance

As mentioned in section 4.2.2, a good reward function must encompass the underlying goal of the agent. As such, if an agent maximizes its cumulative reward, it

should produce increasingly accurate results. To see if the reward function introduced in section 4.2.2 truly satisfies our goal of producing good tractograms, we analyzed the sum of rewards obtained during the hyperparameter search for experiment 2.

To do so, we plotted the sum of rewards obtained by the TD3 agent against the VC ratio and OL produced by the Tractometer, during their last validation run. Because longer trajectories may indicate a greater reward (as the agent receives a reward during more timesteps), we also plotted the length of the produced streamlines for the same validation runs.

4.3.2 Evaluation metrics

To gauge performance against prior methods, the Tractometer [23, 24] tool was used. The Tractometer segments a tractogram into "valid" and "invalid" streamlines according to ground-truth bundles and then extracts performance measures. While it was first used on the FiberCup, it was later on adapted to the ISMRM2015 dataset. The Tractometer reports the true-positive rate (streamlines connecting regions that should be connected) as *valid connections* (VC), the false-positive rate (streamlines connecting two regions that should not be connected) as *invalid connection* (IC) and the ratio of streamlines ending prematurely without connecting two regions as no-connections (NC). The tractometer also reports the number of valid bundles (VB), the number of ground-truth bundles with at least one streamline reconstructed, as well as invalid bundles (IB), the number of regions that are connected by at least one streamline but should not be. To assess the reconstruction of the white-matter volume, the Tractometer also reports mean overlap (OL) and mean overreach (OR), a voxel-wise measure of the overlap (having the same voxels containing streamlines) and overreach (extraneous voxels containing streamlines) between reconstructed and ground-truth bundles.

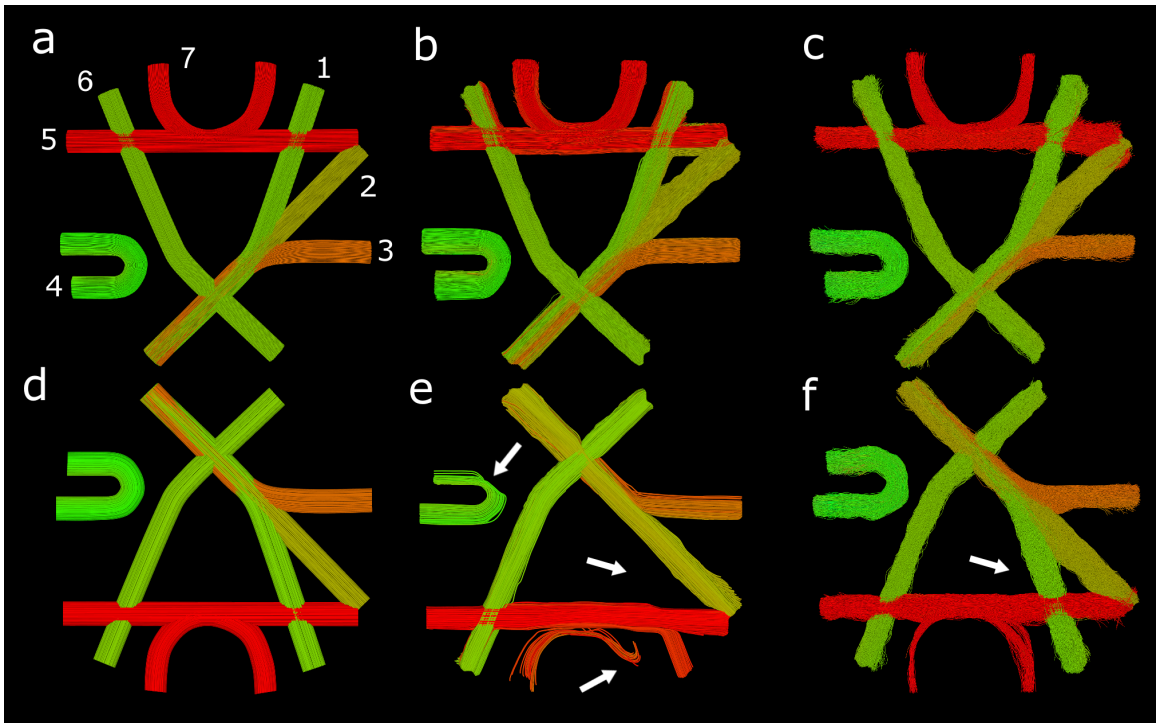

Figure 4.5 – Visualization of the flipped FiberCup dataset and valid connections reconstructed by Learn-to-Track [104] and our method. We can observe the reconstruction done by the Track-To-Learn agent is quite similar if compared between datasets, while the quality of the reconstruction by the supervised method degrades quite significantly. 1-7) Bundle labels. a) Ground-truth FiberCup fibers. b) FiberCup reconstruction by the Learn-to Track (DWI) algorithm. c) FiberCup reconstruction by the SAC agent. d) Ground-truth "Flipped" fibers. e) "Flipped" reconstruction by the Learn-to Track (DWI) algorithm. f) "Flipped" reconstruction by the SAC agent.

Table 4.1 – Results obtained by training/testing on the FiberCup. Testing was also done on a flipped version of it. **Bold** values indicate the best results for each metric and dataset. * indicates Learn-to-Track trained with the same input as the proposed work.

	FiberCup	Flipped	FiberCup	Flipped	FiberCup	Flipped
	VC (% ↑)		VB (↑)		OL (% ↑)	
Ours (TD3)	**76.8**	70.0	**7**	**7**	71.5	75.8
Ours (SAC)	70.0	**71.3**	**7**	**7**	84.1	86.7
CSD-DET	69.1	67.3	**7**	**7**	86.2	84.5
CSD-PROB	24.8	23.8	**7**	**7**	**98.5**	**98.5**
Learn-to-Track	69.6	33.3	**7**	6	95.0	64.1
Learn-to-Track*	73.1	50.5	**7**	**7**	96.9	87.8
	IC (% ↓)		IB (↓)		NC (% ↓)	
Ours (TD3)	19.0	20.5	**1**	**1**	**4.2**	**9.1**
Ours (SAC)	22.5	17.1	**1**	**1**	7.5	11.6
CSD-DET	6.1	**8.7**	**1**	**1**	24.8	23.9
CSD-PROB	14.0	15.5	2	**1**	61.2	60.7
Learn-to-Track	8.7	13.2	**1**	2	21.7	53.6
Learn-to-Track*	**7.2**	9.7	2	2	19.7	39.8

4.4 Results

4.4.1 Experiment 1: Performance on the FiberCup dataset

Table 4.1 shows competitive performances from our agents when training and testing on the same dataset. Compared to the non-machine learning methods CSD-DET and CSD-PROB, our method achieves better VC and NC rates on the original and flipped dataset at the expense of a slightly worse OL and IC. Note that our reported results outperform all of the submissions from the original Tractometer challenge (cf. [24] for more details).

Compared to Learn to track [104], our method reports similar VC rates on the original dataset, but significantly better results on the "Flipped" dataset. While the reported OL is lower than Learn-to-Track on the training dataset, our method has a much better OL on the Flipped dataset. This shows that our method is less prone to overfitting and has better generalization capabilities. Also, although the IC rate is higher for our method, our NC rate is much lower in both settings compared to supervised and classical methods.

Figure 4.5 provides as visual comparison of the reconstructed valid tracks by Learn-to-Track (DWI) [104] and our SAC agent. We can observe that on the flipped dataset (2nd row) that the reconstruction by Learn-to-Track is missing bundle 1 and that it seems to try to "go back" in bundle 7, possibly indicating some over-fitting. While the reconstruction by the proposed method is a bit "noisier", we can appreciate that tractograms from the FiberCup and "Flipped" datasets are very similar. A deeper analysis on the noisiness of the reconstructed streamlines is done in section 4.5.7.

4.4.2 Experiment 2: Performance on the ISMRM2015 dataset

Figure 4.6 provides a visual comparison of the ground-truth bundles against bundles produced by the Track-to-Learn SAC agent. Table 4.2 shows results of our method trained and tested on the ISMRM2015 dataset. We compare our results against the original ISMRM2015 White-Matter Tractography Challenge submissions, as well as state-of-the-art methods by Neher et al. [100, 99], Poulin et al. [104] and Benou et al. [19]. Mean results from the ISMRM2015 challenge were extracted from Poulin et

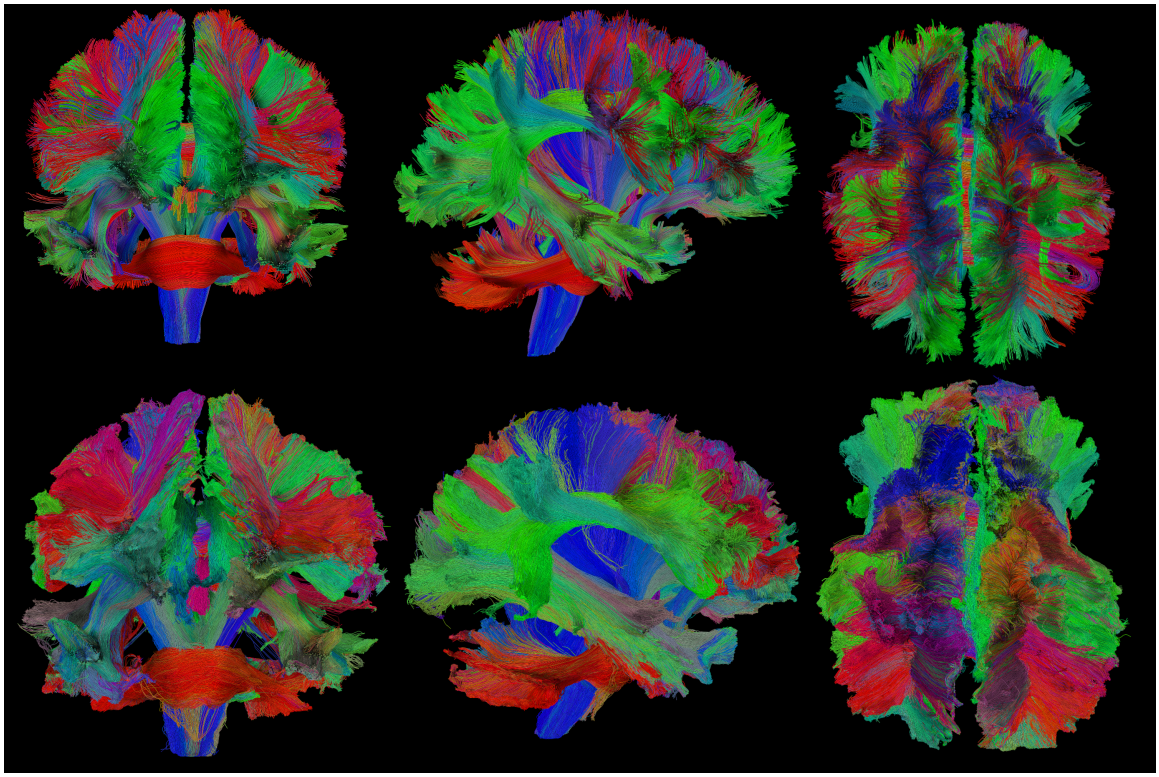

Figure 4.6 – Visualization of the ground-truth ISMRM2015 tractogram and its reconstruction by the proposed method. *Top row*: Ground-truth bundles. *Bottom row*: Reconstruction by the SAC agent.

Table 4.2 – Metrics for our method against the mean results for original ISMRM2015 White Matter Challenge submissions [93] and prior machine learning methods [99, 104, 19]. * indicates methods that were trained using the ground-truth streamlines.'–' indicates metrics that were not reported by the authors of prior methods. **Bold** values indicate the best results for each metric and dataset.

	VC (% ↑)	VB (↑)	OL (% ↑)	IC (% ↓)	IB (↓)	NC (% ↓)
Ours (TD3)	55.1	23	55.8	44.2	163	**0.7**
Ours (SAC)	68.4	23	62.1	30.8	161	**0.7**
ISMRM2015 [93]	53.6	21.4	31.0	19.7	281	25.2
CSD-DET	60.6	23	51.9	33.6	134	5.8
CSD-PROB	33.1	23	**79.3**	53.6	178	13.3
Random Forest	61.1	24.3	55.1	–	86.9	–
DeepTract	40.5	23	34.4	32.6	**51**	22.9
Entrack	65	24	60	–	117	–
Random Forest*	**75.6**	24.3	70.6	–	86.9	–
DeepTract*	70.6	**25.0**	69.3	**19.5**	56	9.9
Learn-to-Track*	41.6	23	64.4	45.6	130	12.8

al. [104]. Because they only report improvement over mean ISMRM2015 results and not direct metrics, performances measures for Neher et al. were re-calculated by the authors based on ISMRM2015 mean results.

Our method is competitive compared to prior methods without the need for labelled bran fibers. While previous ML-based method exhibit high performance when trained on the ground-truth data (which is never available on real *in-vivo* data), we can see a clear drop in their performance when trained on manually segmented tractograms (cf. rows 6-7 vs 8-9). As such, while our method is outperformed on some metrics by algorithms trained on "perfect" ground-truth data, we report competitive results compared to ML algorithms trained on realistic labelled data.

To this extend, we achieve a higher VC rate than Learn-to-Track [104], which was trained on GT data, as well Benou et al.'s DeepTract and Neher et al.'s random forest trained on realistic data. Moreover, we also obtain a higher overlap than supervised methods trained on realistic data. While our reported IC rate is the highest, we also report the lowest NC rate of all methods as well as a similar number of valid bundles as all prior supervised learning methods.

4.4.3 Experiment 3: Generalization through the HCP and ISMRM2015 datasets

In Table 4.3, we compare our method against Learn-to-Track [104] with both the raw diffusion signal and the fODF SH coefficients concatenated with the white-matter mask as input. We also provide results for prior methods. Hyperparameters for this experiment are available in appendix 4.A.3. Overall, we report highly-competitive VC and OL rates, competitive VB and IC rates and the lowest NC rates. We analyze the high number of invalid bundles (IB) produced by our method in section 4.5.4.

4.4.4 Experiment 4: Impact of the discount parameter

Here we analyze he impact of the discount parameter on the (FiberCup) Tractometer metrics. We report some of the final metrics extracted by the Tractometer plotted against the chosen discount for the training runs in figure 4.7.

Table 4.3 – Results obtained after training on the HCP dataset and testing on the ISMRM2015 dataset. Learn-to-Track was trained by the authors and scores for Neher et al. [99], Wegmayr et al. (2018) [152] as well as Wegmayr et al. (2020) [151] were extracted from Wegmayr et al. (2020) [151]. '–' indicates metrics that were not reported by the authors of prior work. **Bold** values indicate the best results for each metric.

	VC (% ↑)	VB (↑)	OL (% ↑)	IC (% ↓)	IB (↓)	NC (% ↓)
Ours (TD3)	53.6	23	58.0	45.2	181	1.1
Ours (SAC)	68.5	23	**65.8**	**30.7**	186	**0.8**
Learn-to-Track	27.5	**24**	56.2	55.8	218	16.7
Learn-to-Track*	49.8	23	65.6	43.1	172	7.1
Random Forest	52	23	59	–	94	–
iFOD3	**72**	23	16	–	**57**	–
Entrack	52	**24**	58	–	123	–

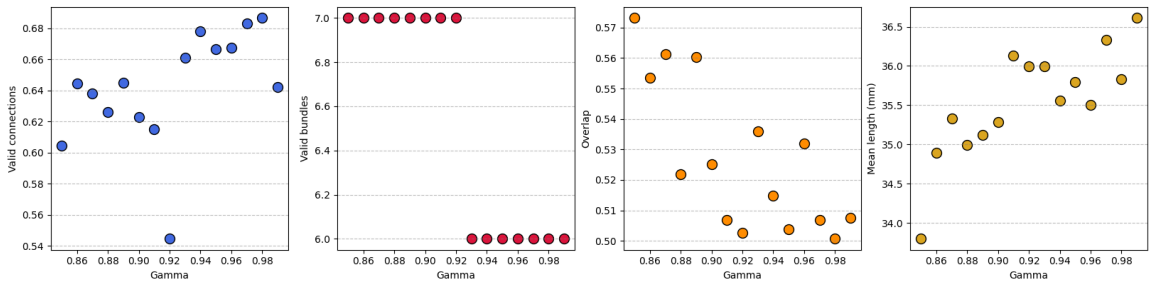

Figure 4.7 – Relation between the discount (γ) parameter and (FiberCup) Tractometer metrics. While the method is quite robust to a wide range of discount values, there seems to be a correlation between the discount factor and the VC, the overlap and the tracks length.

As we can see, our method seems to be robust to a wide variety of discount parameters. We can observe that a higher discount generally leads to a higher VC rate in our experiments, as well as longer streamlines. It makes sense that a lower discount would lead to shorter streamlines as the agent has less incentives to keep tracking as future actions being worth less. We can also observe that a higher discount leads to reduced overlap. As such, even though no catastrophic results can be observed from a "bad" choice of γ, it should nonetheless be chosen with care.

4.4.5 Experiment 5: Impact of the exploration rate

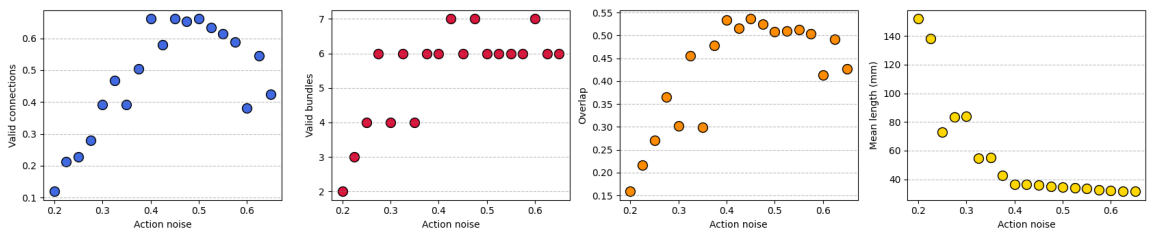

Figure 4.8 – Analysis of the exploration noise versus (FiberCup) Tractometer metrics. We can see that a higher exploration noise generally leads to a higher valid connection rate, but too much noise and the performance decreases.

We report metrics extracted by the (FiberCup) Tractometer against the chosen exploration noise in figure 4.8. As expected, having a higher exploration rate seems to lead towards a higher true-positive rate, more bundles reconstructed and a better overlap, but only up to a certain point. Once the σ_{train} parameter is set too high, the agent cannot properly exploit good actions and the overall performance of the agent decreases.

Too much noise added to the agent's actions forces the agent to an erratic behaviour and more spurious actions, which may drive the streamlines out of the tracking mask prematurely, leading to a reduced mean streamline length. Yet, too little noise might allow the agent to be able to exploit the reward function and track "indefinitely", leading to bad performance, as discussed in section 4.4.7. As such, the exploration rate must be carefully chosen in order to maximize performance.

4.4.6 Experiment 6: Impact of noise at test time

Table 4.4 – Table reporting Tractometer metrics for the same agent with varying levels of noise added during tracking on the ISMRM2015 dataset.

	VC (% ↑)	VB (↑)	OL (% ↑)	IC (% ↓)	IB (↓)	NC (% ↓)
Stochastic	49.2	23	70.4	46.1	200	4.7
0.0	**69.1**	22	51.9	**30.4**	147	**0.5**
0.05	69.0	23	56.5	30.4	141	0.5
0.1	68.4	23	62.1	30.8	161	0.8
0.125	66.8	23	65.1	32.0	166	1.2
0.15	63.4	23	67.8	34.1	175	2.4
0.175	58.0	23	69.9	36.7	178	5.3
0.2	50.6	**24**	**70.6**	39.4	170	10.0
0.225	42.7	23	68.2	41.5	137	15.8
0.25	35.4	23	61.9	42.9	**121**	21.8

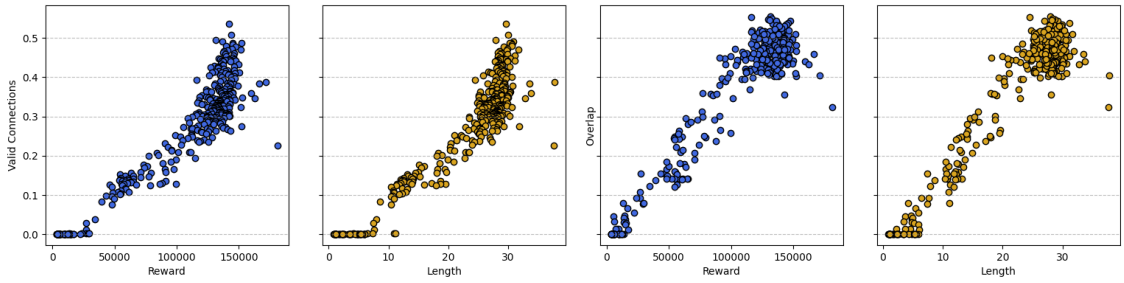

Figure 4.9 – Analysis of the relation between VC and OL (y-axis) versus the final reward obtained by the agent and the mean length (in mm) of the streamlines produced during validation (x-axis). Results come from an hyper-parameter search for Experiment 2. We can see that a higher reward or mean length generally means a higher VC rate and OL up to a point. We also observe a strong correlation between the total reward obtained and the length of the produced streamlines.

In table 4.4, we see that changing the value of σ_{test} allows to control the trade-off between valid connections and overlap, without retraining. With no noise, the VC rate is at its highest but the OL suffers to the point where all bundles may not be reconstructed. As the added noise gets higher, the OL increases but the VC rate

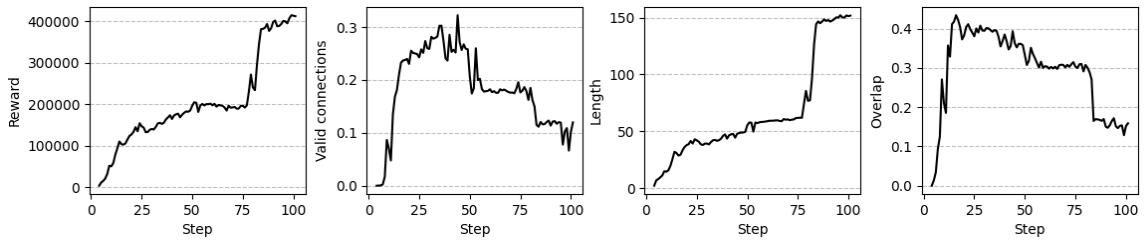

Figure 4.10 – Training curves from an agent achieving high reward on the FiberCup dataset. We can see that a jump in accumulated reward and streamline length around the 80th episode comes with a dip in measured VC rate and OL.

suffers, until too much noise is added and the overall performance degrades. While allowing the agent to control its own noise does seems appealing, we can see that the actions output by the agent are too noisy and lead to poor performance. Interestingly, we can notice the IB rate getting higher alongside σ_{test} until it starts decreasing. We hypothesize that because the NC rate also gets higher, streamlines that would otherwise form invalid bundles are instead forced out of the tracking mask by the noise, ending prematurely and forming NCs instead of IBs.

4.4.7 Experiment 7: Impact of reward on performance

Scatter plots in figure 4.9 show the results of Experiment 7. From the plots, we can infer that achieving a higher cumulative reward does mean better performance, but only up to a certain point. If the received cumulative reward is too high, the performance degrades sharply. We can also see a strong correlation between the reward obtained and the length of the streamlines, which indicates that the agents produce actions that consistently receive a high reward.

4.5 Discussion

4.5.1 Performance

Experiment 1 and 2 show us that despite a simple reward function and architecture, the proposed RL method is able to achieve highly-competitive performance when

compared against classical and SL based methods. From the results presented in sections 4.4.1 and 4.4.2, we can appreciate a high VC rate, high number of reconstructed VB, a decent overlap and a very low rate of NCs.

Looking back at the implementation, and comparing the results against classical methods, we can interpret the Track-To-Learn framework as the groundwork for data-driven improvements over classical algorithms. While the reward function allows the learned agents to behave similarly to classical methods, the controllable level of noise allows the user tracking with Track-to-Learn to directly influence the resulting tractograms. It allows the user to fine-tune the trade-off between probabilistic and classical algorithms, instead of having to choose between the two.

Having a low rate of NCs is very interesting: no-connections indicate streamlines that do not connect two regions. Anatomically speaking, no fibers in the white-matter can have this configuration. As such, these streamlines are usually discarded when performing connectivity analyses, for example, because they most-certainly do not represent the underlying anatomy. While prior methods like Neher et al. [100] had to, for example, implement a hard prior to have streamlines "bouncing off" the white-matter mask to prevent it from terminating prematurely, our agents learned to diverge the tracking process from exiting the white-matter mask on their own.

4.5.2 Generalization capabilities

One of the stronger aspects of the proposed method is certainly its generalization abilities. From experiment 1 and 3, we can observe really strong performance when dealing with a different dataset than the one used for training. Notably, let's take a look at Experiment 1: notice the dip in performance when the prior supervised-learning method tracks on the "flipped" dataset. By looking at figure 4.5, we can see that parts of the reconstructed tractogram seem to try to replicate the original FiberCup instead of reconstructing new, never-seen fibers as would a classical algorithm do. For example, we can see that the streamlines in bundle 7 try to "go back" and follow the direction that would have been correct in the original FiberCup tractogram. Meanwhile, agents trained using the Track-To-Learn framework reconstruct very similar tractograms. Even though the resulting streamlines are noisier (more on that in section 4.5.7), we

can see that all bundles are reconstructed, and that "quirks" learned on the FiberCup dataset are replicated in the flipped version.

Experiment 3 presents competitive results when trained on the HCP dataset, but also reinforces the notion that the algorithm generalizes well to other datasets. Indeed, still comparing the performance presented between experiments 1 and 3, we can notice a drop in performance from other ML methods like Neher et al. [99] and Poulin et al. [104]. However, we report similar performances between the two experiments, implying that our generalization capabilities are not upper bounded by the change in datasets, but rather by the reconstruction capabilities of the proposed method, which could be increased in future works.

4.5.3 Analysis of reward and performance

As demonstrated in 4.4.7, performance degrades when the accumulated reward is too high. To see why performance degrades when the reward is too high, we can take a look at an agent achieving very high reward on the FiberCup. Figure 4.10 shows training curves extracted from an agent achieving high reward. Around the 80th episode, the VC rate drops as the accumulated reward and streamline length blow up. As a comparison, the mean streamline length in the FiberCup's ground-truth set of streamlines is 114mm and the maximum length is of 145mm. Therefore, a final mean length of $\sim$150mm (3rd plot) is too much. A visual inspection of the reconstructed streamlines, available in appendix 4.B, reveals that the agent is able to circumvent streamline termination by "looping", allowing it to track for much longer than it should, therefore accumulating more reward.

Reward hacking [4, 45] happens when the agent finds a way to maximize its accumulated reward while violating the overarching objectives it was supposed to reach. For example, an agent might find a bug in a video game allowing it to achieve a high score without actually progressing in the game and therefore demonstrating knowledge and comprehension of it. By examining the streamlines in appendix 4.B, we can observe that the reward function introduced in section 4.2.2 and the episode-termination conditions introduced in 4.2.2 still allow the agent to gather high reward without producing meaningful tractograms. As such, while the proposed reward

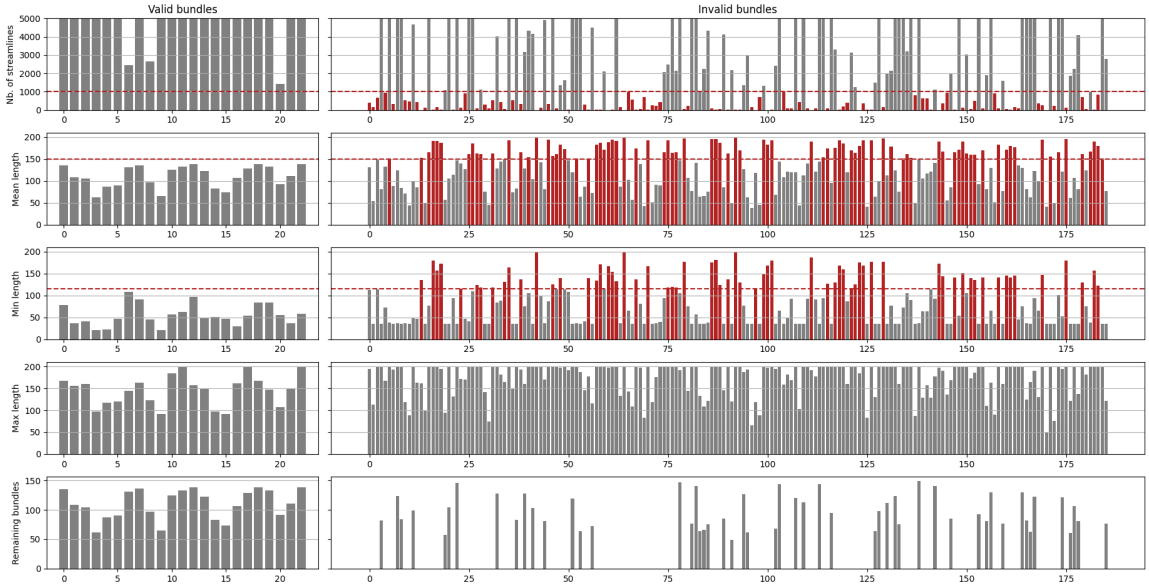

Figure 4.11 – Measures of valid and invalid bundles reconstructed in Experiment 3 by the SAC agent. Y-axis is shared across columns. Red bundles could be discarded according to simple criteria. Left column: measures of valid bundles. Right: measures of invalid bundles. First row: Number of streamlines per bundles, where the y-axis has been cut-off for clarity. Second row: Mean streamline lengths per bundle reconstructed. Third row: Minimum streamline length per bundle reconstructed. Fourth row: Maximum streamline length per bundle reconstructed. Fourth row: Mean length of the 52 (out of 186) remaining bundles after pruning via simple criteria.

function does lead to competitive performance, careful monitoring is still required to make sure it is not taken advantage of by learning agents.

4.5.4 Analysis of invalid bundles reconstructed

Despite good overall results, experiments 2 and 3 display a large number of invalid bundles. Taking into account that the proposed method works by exploring its environment, the fact that it reconstructs a lot of invalid bundles is not very surprising. Figure 4.11 provides some statistics on the reconstructed bundles.

When comparing ourselves to other SL methods, we can observe that our method tends to reconstruct more IBs. This is not surprising if we compare the two learning paradigms: Our approach works by exploring the space of possible fiber trajectories

while SL approaches encounter the "correct" branching options at training time. As stated in section 4.1, local modelling alone, which our method relies upon, cannot help disentangle fiber crossings. We can also observe in experiments 1 and 2 that the proposed method reconstructs a similar number of invalid bundles as the CSD-PROB classical algorithm. Because both our methods receive similar inputs and have stochastic outputs, this is not surprising. Experiment 2 also reveal that our method reconstructs fewer IBs than the average submission of the ISMRM2015 challenge.

Interestingly enough, experiments 1 and 3 show that the proposed work reconstructs fewer or a similar number of IBs than Learn-to-Track when placed in a "new" environment. This may indicate that SL methods may not learn how to disentangle crossings, but merely reconstruct the crossings they have encountered during training. We can also observe an increase in the number of IBs reconstructed for all SL methods between experiments 2 and 3. While iFOD3 reconstructs fewer IBs than the other methods, we see an exceedingly poor overlap, implying that very few streamlines were actually reconstructed, and therefore few streamlines could have taken the wrong path.

The first row of figure 4.11 indicates that the invalid bundles reconstructed by our agents tend to have a very low streamline count. From row 2 and 3, we can observe that invalid bundles tend to have either very long or very short streamlines. Therefore, while a gross filtering of bundles based on their streamline length cannot be considered as it would lead to the elimination of valid bundles, further post-processing of bundles could be done to reduce the number of false-positive. For example, no valid bundle has a mean streamline length above 150mm or a minimum streamline length over 115mm. Row 5 displays the mean length of the 52 (out of 186) bundles that would remain if we were to perform filtering based on the aforementioned criteria.

Therefore, while agents trained via the proposed framework do tend to reconstruct many invalid bundles, we can observe that invalid bundles tend to expose very different characteristics from their valid counterparts and that simple post-processing via bundling and thresholding of the reconstructed streamlines [51] could be used to reduce the number of IBs.

4.5.5 Local modelling and the reward function

Our RL methods use fODFs as part of their input signal and use their peaks as part of their reward function. fODFs are computed by deconvoluting diffusion ODFs [143] with a local fiber model to further "sharpen" it [132]. Subsequently, peaks are computed by extracting the local maxima of the computed fODFs and so, by definition, the fODFs and their peaks are highly correlated. As such, using both of these representations imposes a strong prior on the learning process as the tracking procedure learns to follow the directionality imposed by the input signal. Moreover, the use of local peaks provides the learning agent with a very dense reward signal that will undoubtedly impose a behaviour that is close to classical tractography also using fODF peaks to guide the tracking procedure.

To remove the prior imposed by fODFs and their peaks, inspiration could be drawn from prior work [104, 152] and raw resampled diffusion signal could be used instead of the fODFs as input signal. However, by itself, raw diffusion data does not provide enough directional information to allow for the tracking procedure, and requires at some point some form of processing to present usable directions. Even SL methods which use raw diffusion signal as input depend on modelling methods, as they require labelled data in the form of streamlines, which have to be generated by classical tractography algorithms. As such, while the proposed framework does not completely alleviate the need for signal modelling, it removes the dependency on classical tracking algorithms, and the use of raw diffusion signal as input would further remove priors on the learning process.

Furthermore, the reward function as proposed in section 4.2.2 and discussed in section 4.5.3 forces a learning process that will enforces behaviours that are similar to classical tractography algorithms. To alleviate such constraints, the reward function could be adapted to take into consideration global connectivity by relying on atlases of regions that are known to be connected, possibly disregarding local directionality entirely.

4.5.6 TD3 vs. SAC

While both RL agents achieve competitive results, the SAC agent seems to often outperform TD3.

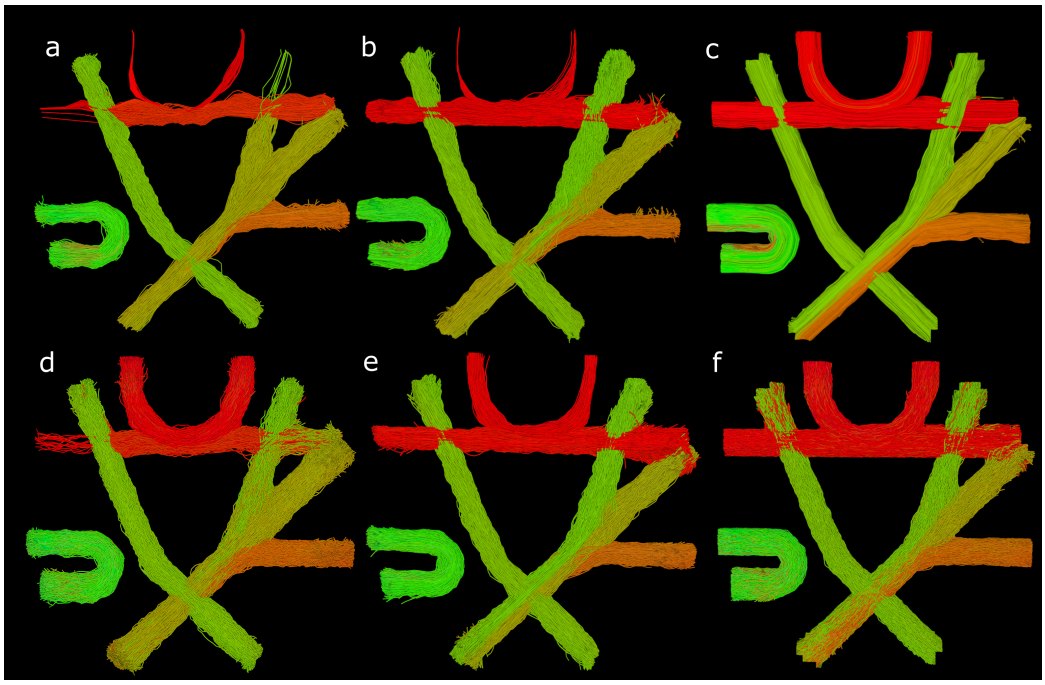

Figure 4.12 – Valid connections reconstructed by Track-to-Learn agents trained for Experiment 1 (with different levels of noise), as well as classical algorithms. a) Reconstruction by the TD3 agent, with $\sigma_{test} = 0$. b) Reconstruction by the SAC agent, with $\sigma_{test} = 0$. c) Reconstruction by a CSD-DET algorithm d) Reconstruction by the TD3 agent, with $\sigma_{test} = 0.1$ e) Reconstruction by the SAC agent, with $\sigma_{test} = 0.1$ f) Reconstruction by a CSD-PROB algorithm.

The TD3 and SAC algorithms share many similarities: both employ two critics predicting a q-value for the actions output by their respective actors. In our implementation, both agents use neural networks with the same number of layers and the same width per layer. However, TD3 employs a deterministic policy, where its output is a vector of three components representing the action. SAC employs a stochastic policy where its output is the mean and standard deviation of a 3D Gaussian distribution, from which actions can be sampled. SAC's policy can be made deterministic by using the predicted mean of the distribution as the actions directly. Moreso, SAC

reformulates the RL objective by including an entropy term, as stated in equation 4.9.

Therefore, both algorithms employ different strategies for exploration. As mentioned in section 4.3.1, TD3 requires explicit noise to be added to the actions output by its deterministic policy. On the opposite, SAC can handle its own exploration by sampling its policy.

We believe it is fair to draw comparisons between the two RL algorithms and classical deterministic and probabilistic algorithms: TD3 outputs direct actions as would a deterministic algorithm, while SAC tries to model the input noise as would a probabilistic algorithm. Adding further to similarities, SAC employs a lower maximum angle between streamline segments than TD3, like would probabilistic algorithms compared to deterministic ones. We determined the maximum angle for each algorithm empirically and found that TD3 would suffer from slow and poor learning with a lower maximum angle, while SAC would produce too many invalid connections and bundles with a higher maximum angle.

The reasons for the discrepancies in results between TD3 and SAC are unclear. Because the two algorithms share so many similarities, we would have expected them to perform similarly in the experiments. Yet, SAC seems to often (but not always) outperform TD3. One hypothesis is the differences in results may come from the entropy bonus of SAC: as the agents encounter crossing regions or bottlenecks, they have to predict different actions depending on their previous directions, while being fed similar observations. As such, TD3 (with its deterministic policy) may have a more natural tendency of taking the same "exit", whereas SAC (being rewarded for having a high entropy) may be able to produce a wider range of actions. However, this is merely an hypothesis and further investigation on the behaviour of agents trained with TD3 and SAC is needed, as well as other algorithms.

4.5.7 Visual assessment of reconstructed streamlines

As can be observed in figure 4.5, the reconstructed streamlines are notably more sinuous than the ground-truth bundles and the reconstruction by Learn-to-Track. This noisiness can be attributed to both the agent themselves and the added stochasticity at test time to promote coverage, as formulated in equation 4.10. Figure 4.12 presents

a comparison of the valid connections reconstructed by Track-to-Learn agents and classical algorithms. As we can observe, and despite having a deterministic output, TD3 seems to reconstruct streamlines that are slightly more spurious than SAC. We hypothesize that the stochastic nature of SAC allows it to better handle the extra noise added at test time. We can also observe an increase in noise as σ_{test} is increased. However, if the noise in the reconstructed streamlines might seem detrimental, especially when compared to the ground-truth bundles, element "f" in figure 4.12 shows that the noise is similar to what would be produced by a probabilistic algorithm. While the reconstruction by the classical algorithms might seem "fuller", the reader should keep in mind the results presented in table 4.1.

Figure 4.6 provides a comparison of the ISMRM2015 dataset ground-truth bundles as well as the reconstruction by the SAC agent of Experiment 2 and allows us to observe some discrepancies between the two tractograms. Notably, we can again observe that the reconstructed streamlines are notably more spurious than the ground-truth bundles, for the reasons mentioned above. We can also see that the streamlines near the cortex don't exactly follow the same direction and seem more "regular" and fuller than the ground-truth, which we can attribute to the differences in white-matter masks used to reconstruct the bundles.

4.5.8 Alternatives to feed-forward neural networks

Initially, and motivated by Poulin et al. [104], we attempted to use recurrent neural-networks instead of their feed-forward counterpart. TD3 uses a replay buffer to store transitions and reuse them for training and so the replay buffer was adapted to also store the memory of the network when it produced the transition. However, with this setup we found the network to be incapable of learning the "directionality" of the streamline. Indeed, the network would learn to map input states to a single action, making going "forward" during tracking possible, but going "backwards" impossible as the network would produce actions as if it was going the other way. As such, recurrent networks were dropped in favour of simple feed-forward networks.

While the exact reason for this incapability of learning the directionality of tracking remains unknown, we made several hypothesis as to why this is the case. First, the

effects of having two environments instead of one as is usual in most RL problems are still yet to be investigated. Because the network has to "re-track" the flipped half-streamlines and new predictions are replaced with the pre-existing segments, the memory of the network might no longer represent a meaningful representation of the signal, preventing the learning process from differencing different directions. Also contrary to other projects where RNNs were used in conjunction with RL [78, 17, 21], only a single learning agent was used instead of multiple. Finally, Kapturowski et al. [78] proposes to have the network predict a couple of "burn-in" steps before truly learning on the transitions to fight saved memory from going "stale", which was not done in our case.

4.5.9 Future works

As mentionned in section 4.5.5, we chose to feed the agent the spherical harmonics coefficients from fiber ODFs instead of the (possibly resampled) raw diffusion signal. This imposes a prior on the data which previous machine learning methods for tractography generally did not have. As such, the proposed method should also be tested using the raw resampled diffusion signal as input, to provide a fairer comparison.

Having two environments makes the present method harder to adapt for reinforcement learning algorithms. Many of them require asynchronous actors learning and having two environments makes this process rather awkward. Therefore, the process of tractography could be adapted to only require one environment by exclusively seeding from the grey-matter/white-matter interface and thus removing the need to go "backwards" as did Wegmayr et al. [152]

As discussed in section 4.5.8, unsuccessful attempts were made to take advantage of the memorial capacities of recurrent neural networks. While appending the last four directions of the streamlines is one way to correct for the directional dependencies of tracks, we believe that more work needs to be done so that RNNs can really be successfully used for tractography.

While our method shares similarities with probabilistic tracking methods, our agents tend to produce "skinnier" bundles with a lower OL than their classical counterparts, as well as a lower overlap than some prior ML methods. While the

reasons for this require more investigation, we can speculate that the agents tend to stay away from the limits of the white-matter mask in fear of terminating. Since the agent is incentivized to track for as long as possible so as to receive a large reward, tracking near the limits of the white-matter is riskier, potentially leading to a lower OL. To alleviate this problem, probabilistic termination maps [57] could be used instead of a binary WM mask to determine streamline termination.

As mentioned in section 4.5.4, the proposed method tends to reconstruct a high number of invalid bundles, as per its exploratory nature and the fact that, reward-wise, invalid bundles are worth as much as valid ones. As such, the reward function must be improved to allow better disentanglement between valid and invalid bundles, probably with an atlas or shape-priors. Because designing a reward function by hand is difficult, an alternative would be to learn the reward function through *inverse reinforcement learning*, where both the policy and the reward function are learned at the same time.

4.6 Conclusion

The ill-posedness of the tractography inverse problem suggests, in our opinion, that reinforcement learning methods are best suited to solve it. RL methods can exploit the expressive capabilities of neural networks without the need for hard-to-obtain and/or biased labelled-data. Therefore, we presented the deep reinforcement learning Track-to-Learn framework. The framework allows for data-driven improvements over classical methods by making a learning agent improve its tracking abilities iteratively via trial-and-error. We report competitive results on a variety of datasets. While prior supervised learning methods struggled with generalization to new, unseen datasets, we report little to no loss of performance when tracking on unknown diffusion volumes. Even with the reported results, the presented method lays the groundwork for tractography via reinforcement learning. There is much room for improvements and our open-source Track-to-Learn framework can used to build, in the future, even better tractography algorithms.

Acknowledgments

The authors would like to thank members of the SCIL [**] and VITAL [††] groups for their suggestions, insight and discussions on this project.

**. http://scil.dinf.usherbrooke.ca/en/
††. http://vital.dinf.usherbrooke.ca/

4.A Experiment hyperparameters

4.A.1 Experiment 1

Tables 4.5, 4.6, 4.7 and 4.8 present the hyperparameters of all agents trained for Experiment 1, organized in tables for clarity.

Table 4.5 – Hyperparameters selected for the TD3 agent of Experiment 1

Hyperparameter	Value
Max. angle	60
Nb. episodes	150
Nb. layers	3
Layer width	1024
Learning rate	9.87e-06
Gamma (γ)	0.80
Exploration noise (σ_{train})	0.32

Table 4.6 – Hyperparameters selected for the SAC agent of Experiment 1

Hyperparameter	Value
Max. angle	30
Nb. episodes	80
Nb. layers	3
Layer width	1024
Learning rate	4.35e-04
Gamma (γ)	0.90
Entropy coefficient (α)	0.087

4.A.2 Experiment 2

Tables 4.9 and 4.10 present the hyperparameters of agents trained for Experiment 2, organized in tables for clarity.

Table 4.7 – Hyperparameters selected for the Learn-to-Track (DWI) agent of Experiment 1

Hyperparameter	Value
Max. angle	30
Nb. epochs	1000
Nb. layers	3
Layer width	1024
Learning rate	1e-3
Patience	20
Weigth Decay	0.001
Gradient clip	50
Dropout	0.1

Table 4.8 – Hyperparameters selected for the Learn-to-Track (fODF + WM Mask) agent of Experiment 1

Hyperparameter	Value
Max. angle	30
Nb. epochs	1000
Nb. layers	3
Layer width	1024
Learning rate	1e-3
Patience	20
Weigth Decay	0.001
Gradient clip threshold	50
Dropout	0.1

4.A.3 Experiment 3

Tables 4.11, 4.12, 4.13 and 4.14 present the hyperparameters of all agents trained for Experiment 3, organized in tables for clarity.

Table 4.9 – Hyperparameters selected for the TD3 agent of Experiment 2

Hyperparameter	Value
Max. angle	60
Nb. episodes	150
Nb. layers	3
Layer width	1024
Learning rate	8.56e-06
Gamma (γ)	0.776
Exploration noise (σ_{train})	0.334

Table 4.10 – Hyperparameters selected for the SAC agent of Experiment 2

Hyperparameter	Value
Max. angle	30
Nb. episodes	100
Nb. layers	3
Layer width	1024
Learning rate	3.70e-05
Gamma (γ)	0.89
Entropy coefficient (α)	0.076

Table 4.11 – Hyperparameters selected for the TD3 agent of Experiment 3

Hyperparameter	Value
Max. angle	60
Nb. episodes	150
Number of layers	3
Width of layers	1024
Learning rate	9.14e-06
Gamma (γ)	0.767
Exploration noise (σ_{train})	0.34

Table 4.12 – Hyperparameters selected for the SAC agent of Experiment 3

Hyperparameter	Value
Max. angle	30
Nb. episodes	100
Number of layers	3
Width of layers	1024
Learning rate	3.45e-04
Gamma (γ)	0.78
Entropy coefficient (α)	0.035

Table 4.13 – Hyperparameters selected for the Learn to Track (DWI) agent of Experiment 3

Hyperparameter	Value
Max. angle	30
Nb. epochs	1000
Nb. layers	3
Layer width	1024
Learning rate	1e-3
Patience	20
Weigth Decay	0.001
Gradient clip threshold	50
Dropout	0.1

4.B Reward hacking

Figure 4.13 provides a visual representation of the behaviour of an agent that "overfits" on the proposed environment.

Table 4.14 – Hyperparameters selected for the Learn to Track (fODF + WM Mask) agent of Experiment 3

Hyperparameter	Value
Max. angle	30
Nb. epochs	1000
Nb. layers	3
Layer width	1024
Learning rate	1e-3
Patience	20
Weigth Decay	0.001
Gradient clip threshold	50
Dropout	0.1

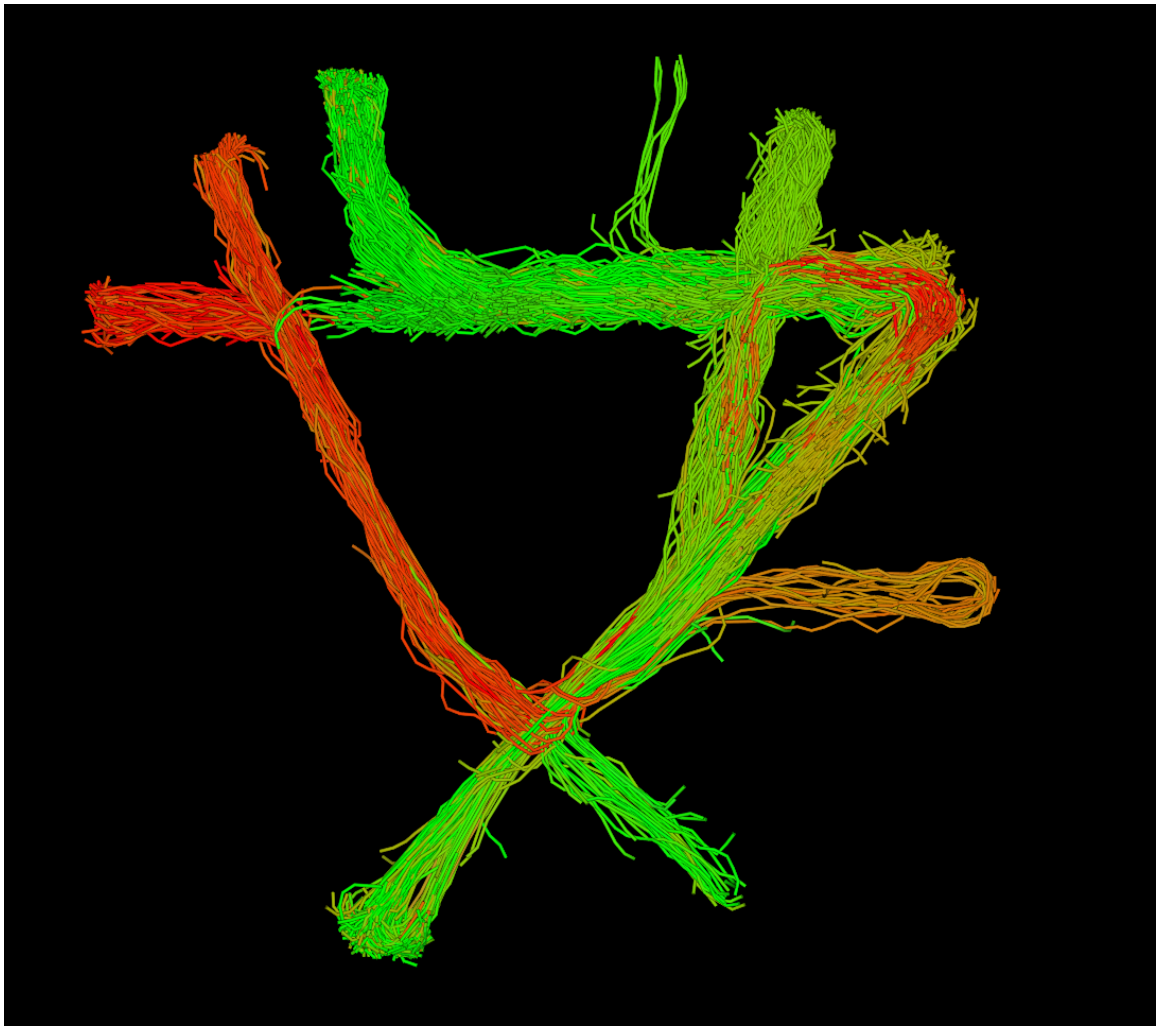

Figure 4.13 – Tractogram reconstructed on the FiberCup phantom, with the shortest streamlines removed. As we can observe, even with the constraints listed in section 4.2.2, the agent might still exploit the environment to track for much longer than it should, performing loops instead of terminating.

It is a puzzling thing. The truth knocks on the door and you say, "Go away, I'm looking for the truth," and so it goes away. Puzzling.

—Robert M. Pirsig, Zen and the Art of Motorcycle Maintenance : An Inquiry Into Values

Conclusion et perspectives

Conclusion

Les avancées dans la modélisation des données de diffusion et l'implémentation d'*a priori* sophistiqués ont permis la conception d'algorithmes de tractographie hautement efficaces et précis. Malgré tout, plusieurs problèmes demeurent avec ceux-ci, notamment la difficulté de conception des a priori. Afin d'apprendre des algorithmes de tractographie hautement performant ne nécessitant pas la conception d'*a priori* complexes, des approches par apprentissage automatique supervisé ont été mises de l'avant. Par contre, ceux-ci dépendent d'une vérité terrain difficile à obtenir et/ou inhéremment biaisée, de par la nature *in vivo* de la tractographie et les problèmes connus avec les algorithmes de tractographie classiques.

Afin d'aborder ces limitations, nous avons introduit une nouvelle classe d'algorithmes de tractographie, appris par apprentissage par renforcement profond. Ces algorithmes permettent d'exploiter la puissance des réseaux de neurones, sans dépendre d'un jeu de données dit *vérité-terrain*. En effet, les réseaux de neurones utilisés à travers le cadre de l'apprentissage supervisé nécessitent habituellement des bases de données annotées afin d'apprendre leur fonction objectif. Par contre, le cadre de l'apprentissage profond laisse les réseaux de neurones amasser eux-même les données nécessaires à leur apprentissage et ne nécessitent qu'une fonction récompense, indiquant la désirabilité des données amassées.

De ce fait, nous avons formalisé le contexte de la tractographie à travers le cadre de l'apprentissage par renforcement profond. Plus spécifiquement :

1. Nous avons proposé *Track-to-Learn*, un cadre d'application général permettant d'entraîner des agents à apprendre des algorithmes de tractographie.

2. Nous avons effectué une analyse de plusieurs composantes du cadre d'application proposé afin d'évaluer leurs impacts sur les tractogrammes reconstruits ainsi que guider les futurs travaux sur la tractographie par apprentissage par renforcement.
3. Nous avons montré des résultats compétitifs lorsque comparé à des algorithmes de tractographie classiques et par apprentissage supervisé.
4. Nous avons montré des capacités de généralisation compétitives face à de nouveaux jeux de données non rencontrés lors de l'entraînement.

Ce cadre d'application permet à l'utilisateur de poser les données de diffusion en tant qu'environnement dans lequel l'agent peut apprendre à reconstruire des tracts. Cet environnement contient une fonction de récompense permettant de guider le processus d'apprentissage des agents afin qu'ils reconstruisent des tracts d'une façon similaire à ce qu'un algorithme de tractographie classique ferait. Finalement, nous avons implémenté deux agents apprenant afin de les entraîner à produire des tractogrammes correspondant à l'anatomie sous-jacente.

De part nos expérimentations, nous avons démontré que le cadre proposé permet d'apprendre des algorithmes de tractographie obtenant des performances égales à l'état de l'art actuel, mais ayant des capacités de généralisation supérieures aux algorithmes de tractographie par apprentissage automatique précédemment proposés, mais sans dépendance à une vérité-terrain problématique. De plus, de part son désir d'accumuler un plus grand retour, les agents entraînés ont tendance à reconstruire significativement moins de vrai-négatifs que les autres algorithmes de tractographie, sans *a priori* forçant explicitement ce comportement. Par contre, nous avons aussi put observer que les agents entraînés reconstruisent un taux élevé de faux-positifs et que les agents ont tendance à exploiter la fonction de récompense afin de produire des tracts beaucoup trop longues.

Perspectives

Tel que mentionné précédemment, les avancées dans la modélisation des données de diffusion et dans les algorithmes de tractographie ont permis la reconstruction de tractogrammes représentatifs de l'anatomie sous-jacente. Alors qu'un seul type de modélisation de données de diffusion, de technique d'initialisation de tracts, de

propagation de tracts et de réseaux de neurones ont été considérées, il est maintenant possible d'utiliser ou de s'inspirer des avancées orthogonales à celles présentées dans ce mémoire afin de concevoir des algorithmes de tractographie encore plus puissants.

Le processus de conception du cadre *Track-to-Learn* fut parsemé d'embûches qui, nous estimons, valent la peine d'être communiquées, évitant ainsi aux chercheurs désirant se lancer dans l'apprentissage par renforcement appliqué à la tractographie de répéter les même erreurs. Notamment, plusieurs fonctions de récompense ont été testées avant d'en arriver à celle présentée dans l'article *Track-to-Learn*. Certaines étaient basées sur la longueur des tracts, sur le masque de la matière grise, sur l'alignement avec les pics des fODFs et certaines sur une combinaison de ces critères. De plus, les réseaux de neurones récurrents, plutôt que les réseaux de neurones à propagation avant, furent d'abord considérés afin d'apprendre le processus de tractographie. L'espace d'état n'incluait initialement pas les dernières directions prises par l'agent, et un autre algorithme d'apprentissage par renforcement, PPO, était utilisé. Nous sommes d'avis que ces choix d'implémentation, ultimement revus pour la version finale de l'article, valent la peine d'être revisités dans un article subséquent afin d'explorer leur impact et de guider de futurs travaux de recherche.

En ayant introduit dans ce mémoire le cadre de la tractographie par apprentissage par renforcement profond, il nous est maintenant possible d'explorer toutes les possibilités que l'apprentissage par renforcement profond offre. En effet, bien que deux algorithmes ont été implémentés dans le cadre *Track-to-Learn*, une pléthore d'algorithmes existe et il sera pertinent de faire un panorama des avantages et inconvénients de chacun lorsque appliqués au problème de la tractographie. De plus, bien des *branches* d'apprentissage par renforcement existent dans la littérature et seraient potentiellement intéressantes à explorer dans le cadre de la tractographie. Par exemple, dans ce mémoire, seul l'apprentissage par renforcement profond *sans-modèle* a été abordé, où une politique apprend directement l'action optimale à poser en réagissant selon l'état de l'environnement. Par contre, l'apprentissage par renforcement profond *avec modèle* existe aussi, où les dynamiques de l'environnement sont apprises et permettent de planifier d'avance les actions à prendre. Cette distinction permet aux algorithmes d'apprendre bien plus rapidement et grâce à moins d'échantillons. Une autre branche de l'apprentissage par renforcement est celle de l'apprentissage par ren-

forcement distributionnel : alors que les fonctions *états-valeurs classiques* apprennent l'espérance du retour selon la politique, la branche distributionnelle propose plutôt d'apprendre la distribution complète du retour potentiel selon la politique. Cette distinction permet de stabiliser le processus d'apprentissage et a mené l'apprentissage par renforcement distributionnel à faire partie des algorithmes dits état-de-l'art dans le contexte des jeux vidéos.

L'apprentissage par renforcement profond nécessite une fonction de récompense afin de guider le processus d'apprentissage de l'agent. La fonction de récompense telle qu'implémentée pour *Track-to-Learn* guide les agents vers des comportements similaires aux algorithmes de tractographie classiques. Hors, ces algorithmes ont des problèmes connus qui ne devraient pas être émulés par les agents apprenants. De ce fait, la fonction de récompense devra être, par exemple, améliorée afin qu'elle puisse permettre aux agents apprenants de ne connecter que les régions qui devraient l'être et d'éviter les *faux-positif*. L'utilisation d'un atlas indiquant quelles paires de régions doivent être connectées pourrait être utilisé dans ce cas.

Par contre, la conception d'une fonction récompense n'est pas chose facile, et ces fonctions sont malheureusement généralement exploitables par les agents apprenants, menant vers le concept de tricherie (*reward hacking*). Parallèlement, l'apprentissage par renforcement inverse permet à un agent apprenant d'apprendre à la fois la politique optimale et la fonction de récompense à partir, par exemple, d'une banque de données de paires d'états et d'actions. L'apprentissage par renforcement inverse évite à l'utilisateur de devoir concevoir une fonction de récompense potentiellement exploitable ou sous-optimale, et permet plutôt d'inférer celle-ci en même temps que la politique la maximisant. Ainsi, il serait intéressant d'utiliser l'apprentissage par renforcement inverse afin d'apprendre le processus de tractographie de tractogrammes déjà existants, plutôt que de concevoir une fonction de récompense pour les reproduire.

En conclusion, nous réitérons que, selon notre opinion, la nature même de la tractographie implique que les approches par apprentissage par renforcement sont les mieux adaptées pour faire face au problème, et espérons que les travaux présentés dans ce mémoire inspireront d'autres chercheurs à emboîter le pas.

How does the universe work ? How did life begin ? Where are my keys ? These are fundamental questions worthy of thought.

—Stuart Russell, Human Compatible : Artificial Intelligence and the Problem of Control

Bibliographie

[1] F. Amato *et al.*, « Artificial neural networks in medical diagnosis, » *Journal of Applied Biomedicine*, vol. 11, no. 2, pp. 47 – 58, 2013. Disponible à http://www.sciencedirect.com/science/article/pii/S1214021X14600570

[2] I. Aganj *et al.*, « Reconstruction of the orientation distribution function in single- and multiple-shell q-ball imaging within constant solid angle, » *Magnetic Resonance in Medicine*, vol. 64, no. 2, pp. 554–566, 2010. Disponible à https://onlinelibrary.wiley.com/doi/abs/10.1002/mrm.22365

[3] A. W. Anderson, « Measurement of fiber orientation distributions using high angular resolution diffusion imaging, » *Magnetic Resonance in Medicine : An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 54, no. 5, pp. 1194–1206, 2005.

[4] D. Amodei *et al.*, « Concrete problems in AI safety, » *arXiv preprint arXiv :1606.06565*, 2016.

[5] H. Azadbakht *et al.*, « Validation of high-resolution tractography against in vivo tracing in the macaque visual cortex, » *Cerebral cortex*, vol. 25, no. 11, pp. 4299–4309, 2015.

[6] B. B. Avants, N. Tustison, et G. Song, « Advanced normalization tools (ANTS), » *Insight j*, vol. 2, pp. 1–35, 2009.

[7] T. E. Behrens, H. J. Berg, S. Jbabdi, M. F. Rushworth, et M. W. Woolrich, « Probabilistic diffusion tractography with multiple fibre orientations : What can we gain ? » *Neuroimage*, vol. 34, no. 1, pp. 144–155, 2007.

[8] G. Brockman *et al.*, « Openai gym, » *arXiv preprint arXiv :1606.01540*, 2016.

[9] G. Brockman *et al.*, « OpenAI Gym, » 2016.

[10] R. Bellman, « A Markovian decision process, » *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[11] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol. II*, 3rd édition. Athena Scientific, 2007.

[12] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[13] P. J. Basser, J. Mattiello, et D. LeBihan, « Estimation of the effective self-diffusion tensor from the NMR spin echo, » *Journal of Magnetic Resonance, Series B*, vol. 103, no. 3, pp. 247–254, 1994.

[14] P. J. Basser, J. Mattiello, et D. LeBihan, « MR diffusion tensor spectroscopy and imaging, » *Biophysical journal*, vol. 66, no. 1, pp. 259–267, 1994.

[15] M. G. Bellemare, Y. Naddaf, J. Veness, et M. Bowling, « The arcade learning environment : An evaluation platform for general agents, » *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[16] P. J. Basser et C. Pierpaoli, « Microstructural and Physiological Features of Tissues Elucidated by Quantitative-Diffusion-Tensor MRI, » *Journal of Magnetic Resonance, Series B*, vol. 111, no. 3, pp. 209 – 219, 1996. Disponible à http://www.sciencedirect.com/science/article/pii/S1064186696900862

[17] A. P. Badia *et al.*, « Agent57 : Outperforming the Atari Human Benchmark, » *arXiv :2003.13350 [cs, stat]*, mars 2020, arXiv : 2003.13350. Disponible à http://arxiv.org/abs/2003.13350

[18] P. J. Basser, S. Pajevic, C. Pierpaoli, J. Duda, et A. Aldroubi, « In vivo fiber tractography using DT-MRI data, » *Magnetic Resonance in Medicine*, vol. 44, no. 4, pp. 625–632, 2000. Disponible à https://onlinelibrary.wiley.com/doi/abs/10.1002/1522-2594%28200010%2944%3A4%3C625%3A%3AAID-MRM17%3E3.0.CO%3B2-O

[19] I. Benou et T. R. Raviv, « Deeptract : A probabilistic deep learning framework for white matter fiber tractography, » dans *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2019, pp. 626–635.

[20] E. Bullmore et O. Sporns, « Complex brain networks : graph theoretical analysis of structural and functional systems, » *Nature reviews neuroscience*, vol. 10, no. 3, pp. 186–198, 2009.

[21] A. P. Badia *et al.*, « Never Give Up : Learning Directed Exploration Strategies, » *arXiv :2002.06038 [cs, stat]*, février 2020, arXiv : 2002.06038. Disponible à http://arxiv.org/abs/2002.06038

[22] M. Chamberland, M. Bernier, D. Fortin, K. Whittingstall, et M. Descoteaux, « 3D interactive tractography-informed resting-state fMRI connectivity, » *Frontiers in Neuroscience*, vol. 9, p. 275, 2015. Disponible à https://www.frontiersin.org/article/10.3389/fnins.2015.00275

[23] M.-A. Côté, A. Boré, G. Girard, J.-C. Houde, et M. Descoteaux, « Tractometer : Online Evaluation System for Tractography, » dans *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2012*, série Lecture Notes in Computer Science, N. Ayache, H. Delingette, P. Golland, et K. Mori, éditeurs. Springer, 2012, pp. 699–706.

[24] M.-A. Côté *et al.*, « Tractometer : Towards validation of tractography pipelines, » *Medical Image Analysis*, vol. 17, no. 7, pp. 844–857, 2013. Disponible à http://www.sciencedirect.com/science/article/pii/S1361841513000479

[25] M. Catani, R. J. Howard, S. Pajevic, et D. K. Jones, « Virtual in Vivo Interactive Dissection of White Matter Fasciculi in the Human Brain, » *NeuroImage*, vol. 17, no. 1, pp. 77 – 94, 2002. Disponible à http://www.sciencedirect.com/science/article/pii/S1053811902911365

[26] S. J. Cook *et al.*, « Whole-animal connectomes of both Caenorhabditis elegans sexes, » *Nature*, vol. 571, no. 7763, pp. 63–71, 2019.

[27] C. Cortes et V. Vapnik, « Support-vector networks, » *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[28] K. Cho *et al.*, « Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, » *arXiv :1406.1078 [cs, stat]*, septembre 2014, arXiv : 1406.1078. Disponible à http://arxiv.org/abs/1406.1078

[29] G. Cybenko, « Approximation by superpositions of a sigmoidal function, » *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[30] M. Descoteaux, E. Angelino, S. Fitzgibbons, et R. Deriche, « Regularized, fast, and robust analytical Q-ball imaging, » *Magnetic Resonance in Medicine : An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 58, no. 3, pp. 497–510, 2007.

[31] G. Dulac-Arnold, D. Mankowitz, et T. Hester, « Challenges of real-world reinforcement learning, » *arXiv preprint arXiv :1904.12901*, 2019.

[32] M. Descoteaux, R. Deriche, T. R. Knosche, et A. Anwander, « Deterministic and probabilistic tractography based on complex fibre orientation distributions, » *IEEE transactions on medical imaging*, vol. 28, no. 2, pp. 269–286, 2008.

[33] M. Descoteaux, R. Deriche, T. Knoesche, et A. Anwander, « Deterministic and probabilistic tractography based on complex fibre orientation distributions, » *IEEE Trans Med Imaging*, vol. 28, no. 2, pp. 269–86, 2009.

[34] M. Descoteaux, R. Deriche, D. Le Bihan, J.-F. Mangin, et C. Poupon, « Multiple q-shell diffusion propagator imaging, » *Medical image analysis*, vol. 15, no. 4, pp. 603–621, 2011.

[35] J. Deng *et al.*, « Imagenet : A large-scale hierarchical image database, » dans *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[36] M. Descoteaux, « High Angular Resolution Diffusion MRI : from Local Estimation to Segmentation and Tractography. (IRM de diffusion à

haute résolution angulaire : estimation locale, segmentation et suivi de fibres), » Thèse de doctorat, University of Nice Sophia Antipolis, France, 2008. Disponible à https://tel.archives-ouvertes.fr/tel-00457458

[37] M. Descoteaux, « High Angular Resolution Diffusion Imaging (HARDI), » dans *Wiley Encyclopedia of Electrical and Electronics Engineering*. John Wiley & Sons, Inc., 2015, pp. 1–25. Disponible à http://doi.wiley.com/10.1002/047134608X.W8258

[38] T. Dhollander *et al.*, « Track orientation density imaging (TODI) and track orientation distribution (TOD) based tractography, » *NeuroImage*, vol. 94, pp. 312–336, 2014.

[39] M. Descoteaux et C. Poupon, « Diffusion-weighted MRI, » 2014.

[40] S. Dreyfus, « Richard Bellman on the birth of dynamic programming, » *Operations Research*, vol. 50, no. 1, pp. 48–51, 2002.

[41] F. Dell'Acqua et J.-D. Tournier, « Modelling white matter with spherical deconvolution : How and why ? » *NMR in Biomedicine*, vol. 32, no. 4, p. e3945, août 2018. Disponible à https://doi.org/10.1002/nbm.3945

[42] P. Di Tommaso *et al.*, « Nextflow enables reproducible computational workflows, » *Nature biotechnology*, vol. 35, no. 4, p. 316, 2017.

[43] T. Degris, M. White, et R. S. Sutton, « Off-policy actor-critic, » *arXiv preprint arXiv :1205.4839*, 2012.

[44] A. Einstein, « Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen, » *Annalen der physik*, vol. 4, 1905.

[45] T. Everitt, « Towards Safe Artificial General Intelligence, » Thèse de doctorat, Australian National University, mai 2018.

[46] P. Fillard *et al.*, « Quantitative evaluation of 10 tractography algorithms on a realistic diffusion MR phantom, » *NeuroImage*, vol. 56, no. 1, pp. 220–234, 2011. Disponible à http://www.sciencedirect.com/science/article/pii/S105381191100067X

[47] S. Fujimoto, H. Hoof, et D. Meger, « Addressing Function Approximation Error in Actor-Critic Methods, » dans *International Conference on Machine Learning*, 2018, pp. 1587–1596. Disponible à http://proceedings.mlr.press/v80/fujimoto18a.html

[48] S. Farquharson *et al.*, « White matter fiber tractography : why we need to move beyond DTI, » *Journal of neurosurgery*, vol. 118, no. 6, pp. 1367–1377, 2013.

[49] Q. Fan *et al.*, « MGH–USC Human Connectome Project datasets with ultra-high b-value diffusion MRI, » *Neuroimage*, vol. 124, pp. 1108–1114, 2016.

[50] E. Garyfallidis *et al.*, « Dipy, a library for the analysis of diffusion MRI data, » *Frontiers in Neuroinformatics*, vol. 8, 2014. Disponible à https://www.frontiersin.org/articles/10.3389/fninf.2014.00008/full

[51] E. Garyfallidis, M. Brett, M. M. Correia, G. B. Williams, et I. Nimmo-Smith, « QuickBundles, a Method for Tractography Simplification, » *Front Neurosci*, vol. 6, p. 175, 2012.

[52] E. Garyfallidis *et al.*, « Recognition of white matter bundles using local and global streamline-based registration and clustering, » *NeuroImage*, vol. 170, pp. 283–295, 2018.

[53] A. Graves, S. Fernández, F. Gomez, et J. Schmidhuber, « Connectionist temporal classification : labelling unsegmented sequence data with recurrent neural networks, » dans *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 369–376.

[54] A. Graves, A.-r. Mohamed, et G. Hinton, « Speech recognition with deep recurrent neural networks, » dans *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[55] M. F. Glasser *et al.*, « The Human Connectome Project's neuroimaging approach, » *Nature Neuroscience*, vol. 19, no. 9, pp. 1175–1187, 2016. Disponible à https://www.nature.com/articles/nn.4361

[56] M. F. Glasser *et al.*, « The minimal preprocessing pipelines for the Human Connectome Project, » *NeuroImage*, vol. 80,

pp. 105 – 124, 2013, mapping the Connectome. Disponible à http://www.sciencedirect.com/science/article/pii/S1053811913005053

[57] G. Girard, K. Whittingstall, R. Deriche, et M. Descoteaux, « Towards quantitative connectivity analysis : reducing tractography biases, » *NeuroImage*, vol. 98, pp. 266–278, septembre 2014.

[58] H. V. Hasselt, « Double Q-learning, » dans *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, et A. Culotta, éditeurs. Curran Associates, Inc., 2010, pp. 2613–2621. Disponible à http://papers.nips.cc/paper/3964-double-q-learning.pdf

[59] J. Huang, R. Friedland, et A. Auchus, « Diffusion tensor imaging of normal-appearing white matter in mild cognitive impairment and early Alzheimer disease : preliminary evidence of axonal degeneration in the temporal lobe, » *American journal of neuroradiology*, vol. 28, no. 10, pp. 1943–1948, 2007.

[60] P. Hagmann *et al.*, « Understanding diffusion MR imaging techniques : from scalar diffusion-weighted imaging to diffusion tensor imaging and beyond, » *Radiographics*, vol. 26, no. suppl_1, pp. S205–S223, 2006.

[61] R. Höftberger et H. Lassmann, « Inflammatory demyelinating diseases of the central nervous system, » dans *Handbook of clinical neurology*. Elsevier, 2018, vol. 145, pp. 263–283.

[62] C. P. Hess, P. Mukherjee, E. T. Han, D. Xu, et D. B. Vigneron, « Q-ball reconstruction of multimodal fiber orientations using the spherical harmonic basis, » *Magnetic Resonance in Medicine : An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 56, no. 1, pp. 104–117, 2006.

[63] R. A. Howard, *Dynamic Programming and Markov Processes*. Cambridge, MA : MIT Press, 1960.

[64] S. Hochreiter et J. Schmidhuber, *Long short-term memory*. MIT Press, 1997, vol. 9, no. 8.

[65] T. Haarnoja, H. Tang, P. Abbeel, et S. Levine, « Reinforcement Learning with Deep Energy-Based Policies, » *CoRR*, vol. abs/1702.08165, 2017. Disponible à http://arxiv.org/abs/1702.08165

[66] T. Haarnoja, A. Zhou, P. Abbeel, et S. Levine, « Soft Actor-Critic : Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, » *CoRR*, vol. abs/1801.01290, 2018. Disponible à http://arxiv.org/abs/1801.01290

[67] H. Johansen-Berg et T. E. Behrens, « Just pretty pictures ? What diffusion tractography can add in clinical neuroscience, » *Current opinion in neurology*, vol. 19, no. 4, p. 379, 2006.

[68] M. Jenkinson, C. F. Beckmann, T. E. Behrens, M. W. Woolrich, et S. M. Smith, « Fsl, » *Neuroimage*, vol. 62, no. 2, pp. 782–790, 2012.

[69] B. Jeurissen, M. Descoteaux, S. Mori, et A. Leemans, « Diffusion MRI fiber tractography of the brain, » *NMR in Biomedicine*, vol. 32, no. 4, p. e3785, 2019.

[70] D. K. Jones, M. A. Horsfield, et A. Simmons, « Optimal strategies for measuring diffusion in anisotropic systems by magnetic resonance imaging, » *Magnetic Resonance in Medicine : An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 42, no. 3, pp. 515–525, 1999.

[71] S. Jbabdi et H. Johansen-Berg, « Tractography : Where Do We Go from Here ? » *Brain Connectivity*, vol. 1, no. 3, pp. 169–183, septembre 2011. Disponible à https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3677805/

[72] B. Jeurissen, A. Leemans, J.-D. Tournier, D. K. Jones, et J. Sijbers, « Investigating the prevalence of complex fiber configurations in white matter tissue with diffusion magnetic resonance imaging, » *Human brain mapping*, vol. 34, no. 11, pp. 2747–2766, 2013.

[73] S. Jbabdi, S. N. Sotiropoulos, A. M. Savio, M. Graña, et T. E. Behrens, « Model-based analysis of multishell diffusion MR data for tractography : How to get over fitting problems, » *Magnetic resonance in medicine*, vol. 68, no. 6, pp. 1846–1855, 2012.

[74] A. Krizhevsky *et al.*, « Learning multiple layers of features from tiny images, » University of Toronto, Rapport technique, 2009.

[75] A. Karpathy et L. Fei-Fei, « Deep visual-semantic alignments for generating image descriptions, » dans *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3128–3137.

[76] A. Kendall *et al.*, « Learning to drive in a day, » dans *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8248–8254.

[77] B. W. Kreher, I. Mader, et V. G. Kiselev, « Gibbs tracking : a novel approach for the reconstruction of neuronal pathways, » *Magnetic Resonance in Medicine*, vol. 60, no. 4, pp. 953–963, octobre 2008.

[78] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, et W. Dabney, « Recurrent Experience Replay in Distributed Reinforcement Learning, » dans *International Conference on Learning Representations*, septembre 2018. Disponible à https://openreview.net/forum?id=r1lyTjAqYX

[79] G. M. Kurtzer, V. Sochat, et M. W. Bauer, « Singularity : Scientific containers for mobility of compute, » *PloS one*, vol. 12, no. 5, p. e0177459, 2017.

[80] A. Krizhevsky, I. Sutskever, et G. E. Hinton, « ImageNet Classification with Deep Convolutional Neural Networks, » dans *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, et K. Q. Weinberger, éditeurs, vol. 25. Curran Associates, Inc., 2012, pp. 1097–1105. Disponible à https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[81] D. Le Bihan et E. Breton, « Imagerie de diffusion in vivo par résonance magnétique nucléaire, » *Comptes rendus de l'Académie des sciences. Série 2, Mécanique, Physique, Chimie, Sciences de l'univers, Sciences de la Terre*, vol. 301, no. 15, pp. 1109–1112, 1985.

[82] Y. LeCun, L. Bottou, Y. Bengio, et P. Haffner, « Gradient-based learning applied to document recognition, » *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[83] D. Le Bihan *et al.*, « MR imaging of intravoxel incoherent motions : application to diffusion and perfusion in neurologic disorders. » *Radiology*, vol. 161, no. 2, pp. 401–407, 1986.

[84] D. Le Bihan, R. Turner, P. Douek, et N. Patronas, « Diffusion MR imaging : clinical applications. » *AJR. American journal of roentgenology*, vol. 159, no. 3, pp. 591–599, 1992.

[85] D.-J. Lee, « Nonlinear estimation and multiple sensor fusion using unscented information filtering, » *IEEE Signal Processing Letters*, vol. 15, pp. 861–864, 2008.

[86] F. Latini, M. Hjortberg, H. Aldskogius, et M. Ryttlefors, « The use of a cerebral perfusion and immersion–fixation process for subsequent white matter dissection, » *Journal of neuroscience methods*, vol. 253, pp. 161–169, 2015.

[87] Y. LeCun, P. Haffner, L. Bottou, et Y. Bengio, « Object recognition with gradient-based learning, » dans *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.

[88] T. P. Lillicrap *et al.*, « Continuous control with deep reinforcement learning, » *arXiv preprint arXiv :1509.02971*, 2015.

[89] L.-J. Lin, « Self-improving reactive agents based on reinforcement learning, planning and teaching, » *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[90] C.-P. Lin, V. J. Wedeen, J.-H. Chen, C. Yao, et W.-Y. I. Tseng, « Validation of diffusion spectrum magnetic resonance imaging with manganese-enhanced rat optic tracts and ex vivo phantoms, » *Neuroimage*, vol. 19, no. 3, pp. 482–495, 2003.

[91] S. Mori, B. J. Crain, V. P. Chacko, et P. C. Van Zijl, « Three-dimensional tracking of axonal projections in the brain by magnetic resonance imaging, » *Annals of Neurology : Official Journal of the American Neurological Association and the Child Neurology Society*, vol. 45, no. 2, pp. 265–269, 1999.

[92] J.-F. Mangin *et al.*, « Toward global tractography, » *Neuroimage*, vol. 80, pp. 290–296, 2013.

[93] K. H. Maier-Hein *et al.*, « The challenge of mapping the human connectome based on diffusion tractography, » *Nature communications*, vol. 8, no. 1, pp. 1–13, 2017.

[94] M. Minsky, « Steps toward Artificial Intelligence, » *Proceedings of the IRE*, vol. 49, no. 1, pp. 8–30, Jan 1961.

[95] V. Mnih *et al.*, « Human-level control through deep reinforcement learning, » *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. Disponible à https://www.nature.com/articles/nature14236

[96] E. Mandonnet, S. Sarubbo, et L. Petit, « The nomenclature of human white matter association pathways : proposal for a systematic taxonomic anatomical classification, » *Frontiers in neuroanatomy*, vol. 12, p. 94, 2018.

[97] J. G. Malcolm, M. E. Shenton, et Y. Rathi, « Filtered multitensor tractography, » *IEEE transactions on medical imaging*, vol. 29, no. 9, pp. 1664–1675, 2010.

[98] S. Mori et P. C. Van Zijl, « Fiber tracking : principles and strategies–a technical review, » *NMR in Biomedicine : An International Journal Devoted to the Development and Application of Magnetic Resonance In Vivo*, vol. 15, no. 7-8, pp. 468–480, 2002.

[99] P. F. Neher, M.-A. Côté, J.-C. Houde, M. Descoteaux, et K. H. Maier-Hein, « Fiber tractography using machine learning, » *Neuroimage*, vol. 158, pp. 417–429, 2017.

[100] P. F. Neher, M. Götz, T. Norajitra, C. Weber, et K. H. Maier-Hein, « A machine learning based approach to fiber tractography using classifier voting, » dans *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2015, pp. 45–52.

[101] P. F. Neher, F. B. Laun, B. Stieltjes, et K. H. Maier-Hein, « Fiberfox : Facilitating the creation of realistic white matter software phantoms :

Realistic White Matter Software Phantoms, » *Magnetic Resonance in Medicine*, vol. 72, no. 5, pp. 1460–1470, 2014. Disponible à http://doi.wiley.com/10.1002/mrm.25045

[102] W. Nitz et P. Reimer, « Contrast mechanisms in MR imaging, » *European radiology*, vol. 9, no. 6, pp. 1032–1046, 1999.

[103] M. O'Sullivan, « Leukoaraiosis, » *Practical neurology*, vol. 8, no. 1, pp. 26–38, 2008.

[104] P. Poulin *et al.*, « Learn to Track : Deep Learning for Tractography, » dans *Medical Image Computing and Computer Assisted Intervention - MICCAI 2017*, série Lecture Notes in Computer Science, M. Descoteaux *et al.*, éditeurs. Cham : Springer International Publishing, 2017, pp. 540–547.

[105] A. Paszke *et al.*, « PyTorch : An Imperative Style, High-Performance Deep Learning Library, » dans *Advances in Neural Information Processing Systems 32*, H. Wallach *et al.*, éditeurs. Curran Associates, Inc., 2019, pp. 8024–8035. Disponible à http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[106] P. Poulin, D. Jörgens, P.-M. Jodoin, et M. Descoteaux, « Tractography and machine learning : Current state and open challenges, » *Magnetic Resonance Imaging*, vol. 64, pp. 37–48, 2019. Disponible à http://arxiv.org/abs/1902.05568

[107] Poupon, C., Laribiere, L., Tournier, G., Bernard, J., Fournier, D., Fillard, P., Descoteaux, M., et al. , « A Diffusion Hardware Phantom Looking Like a Coronal Brain Slice, » dans *Proceedings of the International Society for Magnetic Resonance in Medicine, 2010*, 2010.

[108] C. Poupon *et al.*, « New diffusion phantoms dedicated to the study and validation of high-angular-resolution diffusion imaging (HARDI) models, » *Magnetic Resonance in Medicine*, vol. 60, no. 6, pp. 1276–1283, 2008. Disponible à https://onlinelibrary.wiley.com/doi/abs/10.1002/mrm.21789

[109] P. Poulin, F. Rheault, E. St-Onge, P.-M. Jodoin, et M. Descoteaux, « Bundle-Wise Deep Tracker : Learning to track bundle-specific streamline paths, » *Proceedings of the International Society for Magnetic Resonance in medicine ISMRM-ESMRMB*, 2018.

[110] F. Rheault *et al.*, « Tractostorm : The what, why, and how of tractography dissection reproducibility, » *Human Brain Mapping*, vol. 41, no. 7, pp. 1859–1874, 2020. Disponible à https://onlinelibrary.wiley.com/doi/abs/10.1002/hbm.24917

[111] O. Russakovsky *et al.*, « Imagenet large scale visual recognition challenge, » *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[112] D. E. Rumelhart, G. E. Hinton, et R. J. Williams, « Learning representations by back-propagating errors, » *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[113] G. A. Rummery et M. Niranjan, *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, UK, 1994, vol. 37.

[114] F. Rheault, P. Poulin, A. Valcourt Caron, E. St-Onge, et M. Descoteaux, « Common misconceptions, hidden biases and modern challenges of dMRI tractography, » *Journal of Neural Engineering*, vol. 17, no. 1, p. 011001, 2020. Disponible à https://iopscience.iop.org/article/10.1088/1741-2552/ab6aad

[115] C. P. Reddy et Y. Rathi, « Joint multi-fiber NODDI parameter estimation and tractography using the unscented information filter, » *Frontiers in neuroscience*, vol. 10, p. 166, 2016.

[116] F. Rheault *et al.*, « Bundle-specific tractography with incorporated anatomical and orientational priors, » *NeuroImage*, vol. 186, pp. 382–398, 2019.

[117] K. K. Seunarine et D. C. Alexander, « Multiple fibers : beyond the diffusion tensor, » dans *Diffusion Mri*. Elsevier, 2014, pp. 105–123.

[118] R. S. Sutton et A. G. Barto, *Reinforcement Learning : An Introduction*. A Bradford Book, 2018.

[119] K. G. Schilling *et al.*, « Challenges in diffusion MRI tractography – Lessons learned from international benchmark competitions, » *Magnetic Resonance Imaging*, vol. 57, pp. 194–209, 2019. Disponible à http://www.sciencedirect.com/science/article/pii/S0730725X18305162

[120] K. Schilling *et al.*, « Confirmation of a gyral bias in diffusion MRI fiber tractography, » *Human brain mapping*, vol. 39, no. 3, pp. 1449–1466, 2018.

[121] J. Schulman, S. Levine, P. Abbeel, M. Jordan, et P. Moritz, « Trust region policy optimization, » dans *International conference on machine learning*, 2015, pp. 1889–1897.

[122] D. Silver *et al.*, « Deterministic Policy Gradient Algorithms, » dans *Proceedings of the 31st International Conference on Machine Learning*, série Proceedings of Machine Learning Research, E. P. Xing et T. Jebara, éditeurs, vol. 32, no. 1. Bejing, China : PMLR, 22–24 Jun 2014, pp. 387–395. Disponible à http://proceedings.mlr.press/v32/silver14.html

[123] M. R. Sinke *et al.*, « Diffusion MRI-based cortical connectome reconstruction : dependency on tractography procedures and neuroanatomical characteristics, » *Brain Structure and Function*, vol. 223, no. 5, pp. 2269–2285, 2018.

[124] E. St-Onge, A. Daducci, G. Girard, et M. Descoteaux, « Surface-enhanced tractography (SET), » *NeuroImage*, vol. 169, pp. 524–539, avril 2018. Disponible à http://www.sciencedirect.com/science/article/pii/S1053811917310583

[125] P. Sprawls, *Magnetic resonance imaging : principles, methods, and techniques*. Madison, Wis : Medical Physics Pub, 2000.

[126] D. Silver *et al.*, « Mastering the game of Go without human knowledge, » *Nature*, vol. 550, no. 7676, pp. 354–359, 2017. Disponible à https://www.nature.com/articles/nature24270

[127] R. E. Smith, J.-D. Tournier, F. Calamante, et A. Connelly, « Anatomically-constrained tractography : Improved diffusion MRI streamlines tractography through effective use of anatomical information, » *NeuroImage*, vol. 62, no. 3, pp. 1924–1938, 2012. Disponible à https://www.sciencedirect.com/science/article/pii/S1053811912005824

[128] I. Sutskever, O. Vinyals, et Q. V. Le, « Sequence to sequence learning with neural networks, » *Advances in neural information processing systems*, vol. 27, pp. 3104–3112, 2014.

[129] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, et O. Klimov, « Proximal policy optimization algorithms, » *arXiv preprint arXiv :1707.06347*, 2017.

[130] D. S. Tuch *et al.*, « Diffusion MRI of complex tissue structure, » Thèse de doctorat, Massachusetts Institute of Technology, 2002.

[131] J.-D. Tournier, F. Calamante, et A. Connelly, « Robust determination of the fibre orientation distribution in diffusion MRI : Non-negativity constrained super-resolved spherical deconvolution, » *NeuroImage*, vol. 35, no. 4, pp. 1459 – 1472, 2007. Disponible à http://www.sciencedirect.com/science/article/pii/S1053811907001243

[132] J.-D. Tournier, F. Calamante, et A. Connelly, « Robust determination of the fibre orientation distribution in diffusion MRI : non-negativity constrained super-resolved spherical deconvolution, » *NeuroImage*, vol. 35, no. 4, pp. 1459–1472, mai 2007.

[133] J. D. Tournier, F. Calamante, et A. Connelly, « Improved probabilistic streamlines tractography by 2nd order integration over fibre orientation distributions, » dans *Proceedings of the international society for magnetic resonance in medicine*, vol. 1670. Ismrm, 2010.

[134] J.-D. Tournier, F. Calamante, et A. Connelly, « MRtrix : diffusion tractography in crossing fiber regions, » *International journal of imaging systems and technology*, vol. 22, no. 1, pp. 53–66, 2012.

[135] J.-D. Tournier, F. Calamante, D. G. Gadian, et A. Connelly, « Direct estimation of the fiber orientation density function from diffusion-weighted

MRI data using spherical deconvolution, » *NeuroImage*, vol. 23, no. 3, pp. 1176–1185, novembre 2004.

[136] A. Théberge, C. Desrosiers, M. Descoteaux, et P.-M. Jodoin, « Track-To-Learn : A general framework for tractography with deep reinforcement learning, » *bioRxiv*, 2020.

[137] Y. Tassa *et al.*, « Deepmind control suite, » *arXiv preprint arXiv :1801.00690*, 2018.

[138] G. Tesauro, « Temporal Difference Learning and TD-Gammon. » *J. Int. Comput. Games Assoc.*, vol. 18, no. 2, p. 88, 1995. Disponible à http://dblp.uni-trier.de/db/journals/icga/icga18.html#Tesauro95

[139] G. Theaud *et al.*, « TractoFlow : A robust, efficient and reproducible diffusion MRI pipeline leveraging Nextflow & Singularity, » *NeuroImage*, 2020. Disponible à https://www.sciencedirect.com/science/article/pii/S105381192030375X

[140] B. Tunç *et al.*, « Individualized map of white matter pathways : connectivity-based paradigm for neurosurgical planning, » *Neurosurgery*, vol. 79, no. 4, pp. 568–577, 2016.

[141] Tournier, Jacques-Donald and Calamante, F. and Connelly, Alan, « Improved probabilistic streamlines tractography by 2nd order integration over fibre orientation distributions, » *Proc. Intl. Soc. Mag. Reson. Med.*, vol. 18, 2010.

[142] J.-D. Tournier *et al.*, « MRtrix3 : A fast, flexible and open software framework for medical image processing and visualisation, » *bioRxiv*, p. 551739, 2019.

[143] D. S. Tuch, « Q-ball imaging, » *Magnetic Resonance in Medicine : An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 52, no. 6, pp. 1358–1372, 2004.

[144] J. W. Um, « Roles of glial cells in sculpting inhibitory synapses and neural circuits, » *Frontiers in molecular neuroscience*, vol. 10, p. 381, 2017.

[145] O. Vinyals *et al.*, « Grandmaster level in StarCraft II using multi-agent reinforcement learning, » *Nature*, vol. 575, no. 7782, pp. 350–354, 2019. Disponible à https://www.nature.com/articles/s41586-019-1724-z

[146] C. Van Der Malsburg, « Frank Rosenblatt : Principles of Neurodynamics : Perceptrons and the Theory of Brain Mechanisms, » dans *Brain Theory*, G. Palm et A. Aertsen, éditeurs. Berlin, Heidelberg : Springer Berlin Heidelberg, 1986, pp. 245–248.

[147] D. C. Van Essen *et al.*, « The WU-Minn Human Connectome Project : An overview, » *NeuroImage*, vol. 80, pp. 62–79, octobre 2013. Disponible à http://www.sciencedirect.com/science/article/pii/S1053811913005351

[148] D. C. Van Essen et K. Ugurbil, « The future of the human connectome, » *NeuroImage*, vol. 62, no. 2, pp. 1299–1310, août 2012. Disponible à http://www.sciencedirect.com/science/article/pii/S1053811912000493

[149] D. C. Van Essen *et al.*, « The Human Connectome Project : A data acquisition perspective, » *NeuroImage*, vol. 62, no. 4, pp. 2222–2231, octobre 2012. Disponible à http://www.sciencedirect.com/science/article/pii/S1053811912001954

[150] C. J. C. H. Watkins, « Learning from Delayed Rewards, » Thèse de doctorat, King's College, Cambridge, UK, May 1989. Disponible à http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[151] V. Wegmayr et J. M. Buhmann, « Entrack : Probabilistic Spherical Regression with Entropy Regularization for Fiber Tractography, » *International Journal of Computer Vision*, novembre 2020. Disponible à https://doi.org/10.1007/s11263-020-01384-1

[152] V. Wegmayr, G. Giuliari, S. Holdener, et J. Buhmann, « Data-driven fiber tractography with neural networks, » dans *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, 2018, pp. 1030–1033.

[153] T. H. Williams, N. Gluhbegovic, J. Y. Jew, University of Iowa Health Care, et University of Iowa, *The Human brain : dissections of the real brain.* Iowa City, Iowa : University of Iowa, 1997, oCLC : 245534767.

[154] R. J. Williams, « Simple statistical gradient-following algorithms for connectionist reinforcement learning, » *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[155] T. Wanyan, L. Liu, et E. Garyfallidis, « Tractography Using Reinforcement Learning And Adaptive-Expanding Graphs, » dans *International Symposium on Biomedical Imaging*, 2018.

[156] J. Wasserthal, P. Neher, et K. H. Maier-Hein, « TractSeg – Fast and accurate white matter tract segmentation, » *NeuroImage*, vol. 183, pp. 239 – 253, 2018. Disponible à http://www.sciencedirect.com/science/article/pii/S1053811918306864

[157] E. A. Wan et R. Van Der Merwe, « The unscented Kalman filter for nonlinear estimation, » dans *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*. Ieee, 2000, pp. 153–158.

[158] I. Wolf *et al.*, « The medical imaging interaction toolkit (MITK) : a toolkit facilitating the creation of interactive software by extending VTK and ITK, » dans *Medical Imaging 2004 : Visualization, Image-Guided Procedures, and Display*, R. L. G. Jr., éditeur, vol. 5367, International Society for Optics and Photonics. SPIE, 2004, pp. 16 – 27. Disponible à https://doi.org/10.1117/12.535112

[159] S. Yang *et al.*, « A Simplified Crossing Fiber Model in Diffusion Weighted Imaging, » *Frontiers in Neuroscience*, vol. 13, p. 492, 2019.

[160] A. Yendiki *et al.*, « Automated probabilistic reconstruction of white-matter pathways in health and disease using an atlas of the underlying anatomy, » *Frontiers in neuroinformatics*, vol. 5, p. 23, 2011.

[161] K. Yang, K. Qinami, L. Fei-Fei, J. Deng, et O. Russakovsky, « Towards fairer datasets : Filtering and balancing the distribution of the people subtree in the imagenet hierarchy, » dans *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 2020, pp. 547–558.

[162] Y. Zhang, M. Brady, et S. Smith, « Segmentation of brain MR images through a hidden Markov random field model and the expectation-

maximization algorithm, » *IEEE transactions on medical imaging*, vol. 20, no. 1, pp. 45–57, 2001.

[163] H. Zhang, T. Schneider, C. A. Wheeler-Kingshott, et D. C. Alexander, « NODDI : practical in vivo neurite orientation dispersion and density imaging of the human brain, » *Neuroimage*, vol. 61, no. 4, pp. 1000–1016, 2012.

[164] H. Zhu *et al.*, « The Ingredients of Real-World Robotic Reinforcement Learning, » *arXiv preprint arXiv :2004.12570*, 2020.